Improving Prototypical Visual Explanations with Reward Reweighing, Reselection, and Retraining

Aaron J. Li [1] **Robin Netzorg** [2] **Zhihan Cheng** [2] **Zhuoqin Zhang** [2] **Bin Yu** [2]

Abstract

In recent years, work has gone into developing deep interpretable methods for image classification that clearly attributes a model's output to specific features of the data. One such of these methods is the *prototypical part network* (ProtoPNet), which attempts to classify images based on meaningful parts of the input. While this architecture is able to produce visually interpretable classifications, it often learns to classify based on parts of the image that are not semantically meaningful. To address this problem, we propose the *reward reweighing, reselecting, and retraining* (R3) post-processing framework, which performs three additional corrective updates to a pretrained ProtoPNet in an offline and efficient manner. The first two steps involve learning a reward model based on collected human feedback and then aligning the prototypes with human preferences. The final step is retraining, which realigns the base features and the classifier layer of the original model with the updated prototypes. We find that our R3 framework consistently improves both the interpretability and the predictive accuracy of ProtoPNet and its variants.

1. Introduction

With the widespread use of deep learning, making large models interpretable has become an increasingly important goal for the machine learning community. As these models continue to see use in high-stakes situations, practitioners hoping to justify a decision need to understand how a deep model makes a prediction, and trust that those explanations are valuable and correct (Rudin et al., 2021). One such proposed method for image classification is the *prototypical part network* (ProtoPNet), which classifies a given image based on its similarities to prototypical parts of different classes (Chen et al., 2019). This model aims to combine the power of deep learning with an intuitive reasoning module similar to humans.

While ProtoPNet aims to learn meaningful prototypical concepts, in practice, learned prototypes suffer from learning spurious features, such as the background of an image, and inconsistent concepts, such as learning both the head and the wing of a bird (Bontempelli et al., 2023). Problems like these are highly detrimental to the transparency and efficacy of these models, and thus make the models less likely to be utilized by a human user. Various methods have been proposed to improve such questionable visual explanations (Nauta et al., 2021; Barnett et al., 2021; Bontempelli et al., 2023; Huang et al., 2023; Ma et al., 2023), but none of them have attempted to explicitly quantify the human user's preference for the prototypes.

Thus, in addition to improve the model performance itself, the main goal of this work is to prompt the model to produce prototypes that are more aligned with human preferences, which is a crucial step towards model interpretability (Lage et al., 2018). These two objectives also correspond to the *predictive accuracy* and *relevancy* within the well-known predictive,descriptive, relevant (PDR) interpretable machine learning framework (Murdoch et al., 2019).

Towards this end, we propose the *reward reweighing, reselecting, and retraining* (R3) concept-level debugging framework which improves the original ProtoPNet by using a learned reward model to improve the quality of the prototypes. Our method doesn't need to train the model from scratch, and we call the debugged model R3-ProtoPNet. The human feedback R3 requires is a small number of rating data of prototype quality with multiple scales, given by users when they are shown visualizations of image-prototype pairs. With limited human feedback data on the Caltech-UCSD Birds-200-2011 (CUB-200-211) dataset (Welinder et al., 2010), we are able to train a high-quality reward model that achieves 90.1% test accuracy when ranking human prefer-

[1]Harvard University [2]University of California, Berkeley. Correspondence to: Aaron J. Li <jiaxun_li@g.harvard.edu>, Bin Yu <binyu@berkeley.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

The source code of this work is available at https://github.com/aaron-jx-li/R3-ProtoPNet.

ences, serving as a strong measure for prototype quality. Two distinct advantages of having an external reward model that faithfully captures human preferences are:

- The debugging process becomes efficient, because the reward model is pretrained and it doesn't require online feedback on the explanations generated by the current model.
- The metric for prototype quality becomes more maneuverable, as different reward models could capture slightly different user preferences.

We train this reward model from a small pairwise human preference dataset (further explained in sections 3.2 and 3.3). Then, the R3 framework evaluates and updates the prototypes to maximize their induced rewards, and these debugging steps are followed by a retraining step to restore the predictive performance. Empirically, the R3 debugging procedure is able to reduce model's dependence on spurious features and make the visual explanations more favorable to users. When used either individually or as base learners in an ensemble, R3-ProtoPNet outperforms the original ProtoPNet on a held-out test dataset in terms of predictive accuracy. In general, our proposed framework improves upon a class of widely used inherently interpretable deep learning models (i.e. prototype-based models) by efficiently utilizing their own interpretations.

The contributions of this work can be summarized as follows:

- We propose using the learned reward model as a quantified metric of prototypical visual explanation quality and model interpretability
- We introduce the R3 framework and R3-ProtoPNet, which use efficient reward-guided debugging to improve both prototype meaningfulness and predictive performance.

2. Related Work

2.1. Example-based Models and Prototypical Part Networks

There are many explainability and interpretability methods available to the user within the field of interpretable machine learning (Rudin et al., 2021), and the two main goals for the community are (1) to come up with inherently interpretable machine learning paradigms (Agarwal et al., 2022) and (2) to propose reliable explanation methods for model outputs (Ribeiro et al., 2016; Lundberg & Lee, 2017; Murdoch et al., 2018). To ground the discussion, we focus primarily on example-based models, one such example being ProtoPNet. While other example-based methods exist, such as the non-parametric xDNN (Angelov & Soares, 2019) or SITE (Wang & Wang, 2021), which performs predictions directly from interpretable prototypes, we focus on the ProtoPNet due to its intuitive reasoning structure and explicit visual explanations.

Since ProtoPNet's first introduced by Chen et al. (2019), many iterations of follow-up works have been proposed. Work has explored extending the ProtoPNet to different architectures such as transformers (Xue et al., 2022), or sharing class information between prototypes (Rymarczyk et al., 2021). Donnelly et al. (2022) increase the spatial flexibility of ProtoPNet, allowing prototypes to change spatial positions depending on the pose information available in the image. ProtoPNets and variations have seen success in high-stakes applications, such as kidney stone identification (Flores-Araiza et al., 2022) and mammography (Barnett et al., 2021).

Many works have also worked on addressing the original ProtoPNet's overemphasis on spurious features. Nauta et al. (2021) introduce an explainability interface to ProtoPNet, allowing users to see the dependence of the prototype on certain image attributes. Barnett et al. (2021) introduce a variation of the ProtoPNet, IAIA-BL, which biases prototypes towards expert labelled annotations of classification-relevant parts of the image. Other works such as Huang et al. (2023) and Ma et al. (2023) incorporate new modules and constraints into ProtoPNet to improve its empirical performance without using human feedback.

Similar to how we provide human feedback at the interpretation level, Bontempelli et al. (2023) introduce ProtoPDebug, which first asks for binary user feedback on prototypes as "forbidden" or "valid", and then uses a fine-tuning step that includes the collected feedback as a supervised constraint into the ProtoPNet loss function.

Compared with previous approaches, our R3 framework allows users to efficiently collect high-quality human feedback data and train a robust reward model that could be used to both evaluate and debug the original prototypes.

2.2. Learning from Human Feedback

As the term interpretability lacks a mathematical quantification, practitioners have argued that evaluating it well requires human feedback (Doshi-Velez & Kim, 2017). Our method starts from learning a reward model from human feedback, and then use it as an interpretability measure.

Since the success of InstructGPT (Ouyang et al., 2022), Reinforcement Learning from Human Feedback (RLHF) has attracted a lot of attention. But before that, incorporating human feedback into reinforcement learning methods via a learned reward model also has a deep history in reward learning (Christiano et al., 2017; Jeon et al., 2020). Some prior

works incorporate the reward function as a way to weigh the likelihood term (Stiennon et al., 2022; Ziegler et al., 2019). While works taking inspiration from InstructGPT have used proximal policy optimization (PPO) to fine-tune networks with human feedback (Bai et al., 2022), it is unclear to the extent that formal reinforcement learning is necessary to improve models via learned reward functions (Lee et al., 2023), or if the human feedback needs to follow a particular form (Askell et al., 2021).

Different from RLHF, our work doesn't rely on any formal RL algorithms, and instead simply uses reward values as a supervisory signal to guide the search of semantically meaningful prototypes.

3. Reward Reweighed, Reselected, and Retrained Prototypical Part Network (R3-ProtoPNet)

In this section, we first describe the basics of ProtoPNet (Chen et al., 2019), and then present our R3 debugging framework in detail, which includes the collection of high-quality human feedback data, our reward model, and the incorporation of the reward model into debugging via a three-step update procedure.

Algorithm 1 Reward Reweighed, Reselected, and Retrained Prototypical Part Network (R3-ProtoPNet)

1: **Initialize:** Collect high-quality human feedback data and train a reward model.
2: **Reward Reweighing:** Perform the reward-reweighed update for the ProtoPNet, defined in Equation 1. Optimize the loss function, which leads to locally maximal solutions, improving the prototypes.
3: **Prototype Reselection:** Run the reselection procedure based on a reward threshold.
If $\frac{1}{n_k}\sum_{i\in I(p_j)} r(x_i, p_j) < \alpha$, reselect the prototype by sampling from patch candidates and temporarily setting the prototype to a new candidate that passes the acceptance threshold and is unique from other current prototypes.
4: **Retraining:** Retrain the model with the same loss function used in the original ProtoPNet update, to realign the prototypes and the rest of the model.

3.1. Preliminaries on ProtoPNet

Here we adopt the notation used in Chen et al. (2019). The ProtoPNet architecture builds on a base convolutional neural network f, which is then followed by a prototype layer denoted g_p, and a fully connected layer h. Typically, the convolutional features are taken pretrained models like VGG-19, ResNet-34, or DenseNet-121.

The ProtoPNet injects interpretability into these convolutional architectures with the prototype layer g_p, consisting of m prototypes $\boldsymbol{P} = \{p_j\}_{j=1}^m$ typically of size $1 \times 1 \times D$, where D is the shape of the convolutional output $f(x)$. By keeping the depth the same as the output of the convolutional layer, but restricting the height and width to be smaller than that of the convolutional output, the learned prototypes select a patch of the convolutional output. Reversing the convolution leads to recovering a prototypical patch of the original input image x. Using upsampling, the method constructs an activation pattern per prototype p_j.

To use the prototypes to make a classification given a convolutional output $z = f(x)$, ProtoPNet's prototype layer computes a max pooling over similarity scores: $g_{p_j}(z) = \max_{\tilde{z}\in\text{patches}(z)} \log((\|\tilde{z} - p_j\|_2^2 + 1)(\|\tilde{z} - p_j\|_2^2 + \epsilon))$, for some small $\epsilon < 1$. This function is monotonically decreasing with respect to the distance, with small values of $\|\tilde{z} - p_j\|_2^2$ resulting in a large similarity score $g_{p_j}(z)$. Assigning m_k prototypes for all K classes, such that $\sum_{k=1}^K m_k = m$, the prototype layer outputs a vector of similarity scores that matches parts of the latent representation z to prototypical patches across all classes. The final layer in the model is a linear layer connecting similarities to class predictions.

In order to ensure that the prototypes match specific parts of training images, during training the prototype vectors are projected onto the closest patch in the training set. For the final trained ProtoPNet, every p_j corresponds to some patch of a particular image.

3.2. Human Feedback Collection

As mentioned earlier, while ProtoPNet is capable of providing interpretable classifications, the naive training described in (Chen et al., 2019) results in prototypes that focus on spurious and inconsistent features (Barnett et al., 2021; Bontempelli et al., 2023).

A crucial aspect behind the success of learning faithful reward models is the collection of high quality human feedback data. Unclear or homogeneous feedback may result in a poor performing reward model (Christiano et al., 2017). The design of human feedback collection is vitally important to the training of a useful reward model.

The inherent interpretability of ProtoPNet is particularly useful for reward learning. Given a trained ProtoPNet, it is possible for a user, who doesn't have to be an expert, to directly critique the learned prototypes. In the case of classifying birds in the CUB-200-2011 dataset, it is clear that if a prototype gives too much weight to the background of the image (spurious), or if the prototype corresponds to different parts of the bird when looking at different images (inconsistent). Given these prototypes that fail to contribute

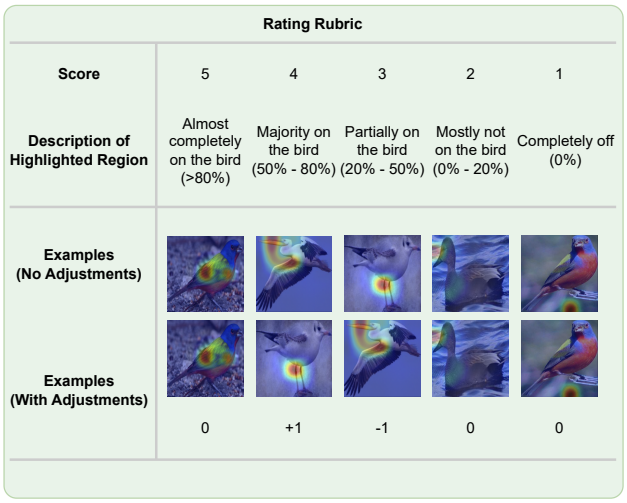

Figure 1. Rubric used for human feedback on the activation patterns of predictions for birds from the CUB-200-2011 dataset. First, the rater estimates a base score based on overlap proportion, and then an optional adjustment $\delta \in \{-1, 1\}$ could be given based on how meaningful or characteristic the focused body part is.

to prediction, a lay person trying to classify birds would be able to rate these prototypes as "bad", with a proper rating rubric.

There are many different ways to elicit the preference of a user (Askell et al., 2021). Although it is possible to incorporate many different forms of feedback into the R3-ProtoPNet, such as asking a user to compare prototypes to elicit preferences or ask for a binary value of whether a prototype is "good" or "bad", we found most success with asking the users to rate a prototype on a scale from 1 to 5. While scalar ratings can be unstable across different raters, with a clear, rule-based rating method, rating variance is reduced and it is possible to generate high-quality labels. An example rating scale on the CUB-200-2011 dataset is provided in Figure 1.

3.3. Reward Learning

We note that, when a user provides feedback on a prototype, it is not the training image or the model prediction that the user is providing feedback on, but the prototype's resulting interpretation: the activation patterns. Our task is therefore different from RLHF applied to language modeling or RL tasks (Ouyang et al., 2022; Christiano et al., 2017), where human feedback is provided on the model output or resulting state. We therefore collect a rating dataset $\mathcal{D} = \{(x_i, y_i, h_{i,j}, r_{i,j})\}_{i=1,j=1}^{n,m}$, where x_i, y_i are the training image and label, and $h_{i,j}, r_{i,j}$ are prototype p_j's activation patterns and user-provided ratings for image x_i. We note that collecting preferences for this entire dataset is prohibitive and unnecessary, so we only collect a subset.

Given the dataset $\mathcal{D}$, we generate the induced comparison dataset, whereby each entry in $\mathcal{D}$ is paired with one another. Given $i \neq i'$ and/or $j \neq j'$, we populate a new paired dataset, $\mathcal{D}_{paired}$, which consists of the entries of $\mathcal{D}$ indexed by i, j, i', j', and a comparison c, which takes values $-1, 1$. If the left-hand sample is greater, and therefore considered higher-quality, $r_{i,j} > r_{i',j'}$, then $c = -1$. If the right-hand sample is greater $r_{i,j} < r_{i',j'}$, then $c = 1$. This synthetic construction allows us to model the reward function, $r(x_i, h_{i,j})$ via the Bradley-Terry Model (Bradley & Terry, 1952), which has demonstrated success in learning pairwise user preferences (Christiano et al., 2017). We train this model with the same loss function as in Christiano et al. (2017), a cross-entropy loss over the probabilities of ranking one pair over the other (See Appendix A for details). This synthetic construction combinatorially increases the amount of preference data, allowing us to train a high-quality reward model on relatively small amounts of human feedback data.

3.4. Reward Reweighed, Reselected, and Retrained Prototypical Part Network (R3-ProtoPNet)

After having collected high-quality human feedback data and trained a reward model, we can now incorporate it into a fine-tuning framework to improve the interpretability of ProtoPNet. We incorporate the reward model via a three step process consisting of reward weighing, reselection, and retraining. Each step is described in more detail below. Figure 2 provides an overview of our proposed framework.

3.4.1. Reward Reweighing

Although PPO is a popular option for RLHF (Ouyang et al., 2022), there is evidence that simpler fine-tuning algorithms can lead to similar performance increases (Askell et al., 2021). Inspired by the success and the efficiency of reward-weighted learning (Lee et al., 2023; Stiennon et al., 2022; Ziegler et al., 2019), we develop a straightforward reward-weighted update for the ProtoPNet:

$$\max_{p_j} \mathcal{L}_{reweigh}(z_i^*, p_j) = \max_{p_j} \sum_{i \in I(p_j)}^{n} \frac{r(x_i, p_j)}{\lambda_{dist} \|z_i^* - p_j\|_2^2 + 1} \tag{1}$$

where $z_i^* = \text{argmin}_{z \in \text{patches}(f(x_i))} \|z - p_j\|_2^2$, $I(p_j) = \{i \mid y_i \in \text{class}(p_j)\}$, and λ_{dist} is a fixed hyperparameter. We note that the objective function $\mathcal{L}_{reweigh}$ is a sum of the inverse distances weighted by the reward of the prototype on that image. Since we only update the prototype p_j, the only way to maximize this objective is to minimize the distance between prototype and image patches with high reward $r(x_i, p_j)$. This causes the prototype to resemble

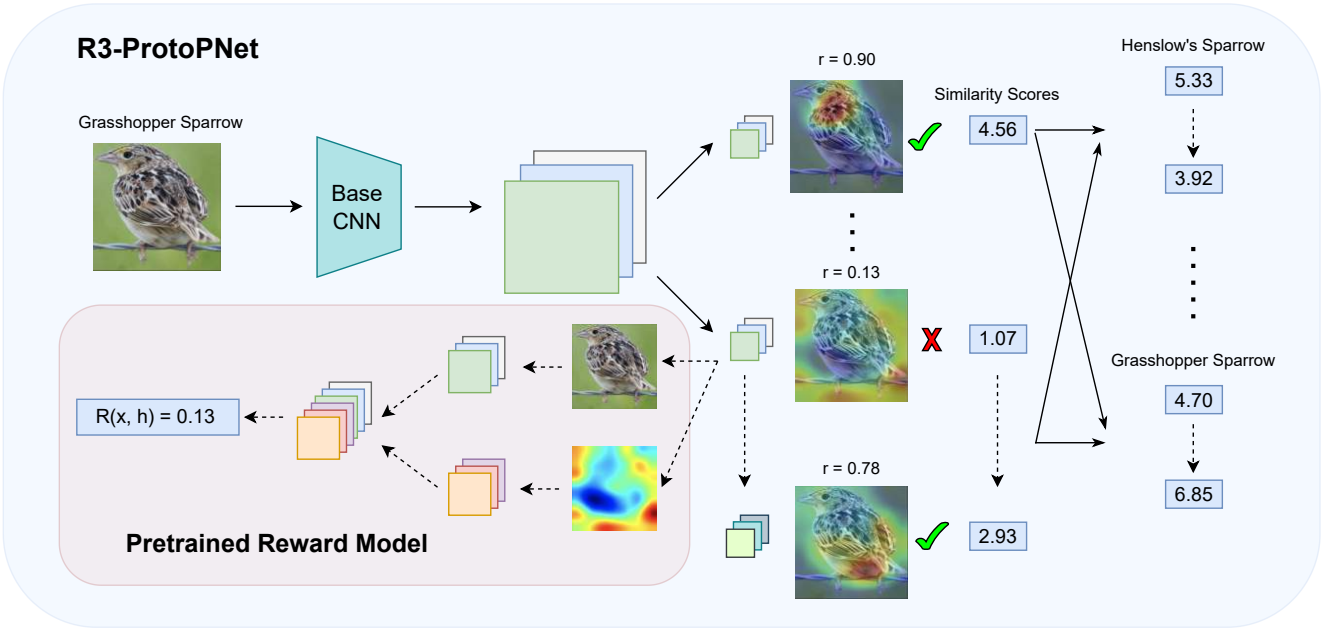

Figure 2. An overview of the R3-ProtoPNet framework. Dashed arrows indicate our R3 debugging procedure.

high reward image patches, improving the overall quality of the prototypes. Wanting to preserve prototypes that already have high reward, we only update those prototypes that have relatively low mean reward γ, and choose this reweighing threshold per base architecture and reward model (see the Appendix for threshold choices). λ_{dist} is included in the objective function to rescale distances, since the closest distances are near zero. We find best performance with $\lambda_{dist} = 100$.

Practically, we find that optimizing this objective function leads to locally maximal solutions, resulting in local updates that do not modify prototypes with low quality values of 1, but it's more likely to improve prototypes with quality values of 2 or higher. If the prototype p_j has high activation over the background of an image x_i, for example, the closest patches z_i^* in the training data will also be background patches, and the reward of the prototype will be low, leaving minimal room for change. In other words, reward reweighing moves the prototype gradually toward better locations on the reward manifold, but it is not possible to dramatically change the location of the illuminated patch via this loss function.

3.4.2. Prototype Reselection

In order to improve low quality prototypes that require significant manipulation, we introduce a reselection procedure based on a reward threshold. Given a prototype p_j and image x_i, if $\frac{1}{n_k}\sum_{i \in I(p_j)} r(x_i, p_j) < \alpha$, where α is a predetermined threshold and n_k is the number of training images in class k, or if the patch of the given prototype p_j matches the patch of another prototype of the same class, we reselect the prototype. The reselection process involves iterating over patch candidates z_i' and temporarily setting the prototype $p_j' = z_i'$, where z_i' is chosen randomly from the patches of a randomly selected image x_i' in the class of p_j. If $\frac{1}{n_k}\sum_{i \in I(p_j)} r(x_i', p_j') > \beta$, where β is an acceptance threshold, and if none of the prototypes match patch $p_j' = z_j'$, then we accept the patch candidate as the new prototype. We found that varying the α and β values per base architecture led to the best performance (See Appendix B for threshold choices). We refer to the combination of reweighing and reselection as the R2 update step, and the corresponding trained model the R2-ProtoPNet.

The reasoning process behind our prototype reselection method takes inspiration from the original push operation in Chen et al. (2019). Similar to how ProtoPNet projects prototypes onto a specific training image patch, here we reselect prototypes to be a particular reward-filtered training image patch. With a high enough acceptance threshold β, this forces the elimination of low reward prototypes while preserving the information gain of having an additional prototype.

One possible alternative approach is to instead search over the training patches and select those patches with the high-

est reward, but we found that randomly selecting patches, in place of an exhaustive search, led to higher prototype diversity and less computation time.

While we do not use a traditional reinforcement learning algorithm to fine-tune our model as is typically done in RLHF (Askell et al., 2021), pairing the reselection/reward-reweighing (R2 update) and retraining steps together resembles the typical explore-exploit trade-off in RL problems: the R2 update serves as a form of exploration, drastically increasing the quality of uninformative prototypes by breaking their dependence on spurious features, while retraining with the updated prototypes resembles exploit behavior, improving upon already high-quality prototypes.

3.4.3. Retraining

A critical step missing in the R2 update is a connection to prediction accuracy. As discussed in Section 4, without incorporating predictive information, performing the reward update alone results in lowered test accuracy. Since the above updates only act on the prototypes themselves, not the rest of the network, the result is a misalignment between the prototypes and the model's base features and final classifier layer. The reward update guides the model towards more interpretable prototypes, but the reward update alone fails to use the higher quality prototypes for better prediction.

To account for the lack of predictive performance, the final step of R3-ProtoPNet is retraining. With the updated prototypes, simply retraining using the same loss function used in the original ProtoPNet training results in the realignment of the prototypes and the rest of the model. Although one could worry that predictive accuracy would reduce the interpretability of the model (Rudin et al., 2021), we find that retraining increases predictive accuracy while maintaining the quality increases of the R2 update. The result is a high accuracy model with higher-quality prototypes. We explore evidence of this phenomenon and why this is the case in the following section.

4. Experiments

4.1. Bird Species Identification

Here we discuss the results of training the R3-ProtoPNet on the CUB-200-2011 dataset, the same dataset as used in Chen et al. (2019).

4.1.1. Data Preprocessing

R3-ProtoPNet requires the original dataset for the initial training, as well as additional scalar ratings of the selected activation patterns produced by image-prototype pairs. Combined, this results in the dataset described in Section 3. To offer better comparison against the original ProtoPNet, we use the same dataset for initial training that was used in Chen et al. (2019), the CUB-200-2011 dataset (Wah et al., 2011). The CUB-200-2011 dataset consists of roughly 30 images of 200 different bird species. We employ the same data augmentation scheme used in Chen et al. (2019), which adds additional training data by applying a collection of rotation, sheer, and skew perturbations to the images, resulting in a larger augmented dataset.

For the collection of the activation pattern ratings, we only provided the activation patterns overlaid on the original images to the raters. Using Amazon Mechanical Turk to recruit six workers per prototype-image pair, we take the average as the user-provided rating for that pair. We also exclude the entries with $|r_{i,j} - r_{i',j'}| < 0.5$ to increase the contrast between pairs. In total, 700 rated prototype-image pairs are collected according to the scale approach described in Figure 1, and we randomly selected 500 of them (the rest were used as a held-out test set to evaluate the robustness) to train the reward model.

4.1.2. Implementation

Similar to Chen et al. (2019), we study the performance of R3-ProtoPNet across five different base architectures: VGG-19, ResNet-34, ResNet-50, DenseNet-121, and DenseNet-161. While the original ProtoPNet sets the number of prototypes per class at $m_k = 10$, we additionally run the VGG-19 architecture with $m_k = 5$ prototypes to explore model performance when the number of prototypes is limited. No other modifications were made to the original ProtoPNet architecture. At most 50 epochs are needed in this initial training step.

The reward model $r(x_i, h_i)$ is similar to the base architecture of the ProtoPNet. Two ResNet-50 base CNNs take in the input image x_i and the associated acticvation pattern h_i separately, and both have two additional convolutional layers. The outputs of the convolutional layers are concatenated and fed into a final linear layer with sigmoid activation to predict the Bradley-Terry ranking. Predicted rewards are therefore bound in the range $(0, 1)$. We train the reward model for 5 epochs on a synthetic comparison dataset of 49K paired images and preference labels derived from 500 human ratings, and evaluate on 14K testing pairs. The reward model achieves 90.09% test accuracy. We additionally analyze the sensitivity of the reward model and R3-ProtoPNet to the amount of human feedback used for reward model training (see Appendix C), and the results suggest that the performance gain of R3-ProtoPNet can be achieved with even fewer human ratings (around 300 image-prototype pairs).

4.1.3. Evaluation Metrics

To evaluate the performance of R3-ProtoPNet, we compare it to ProtoPNet on three metrics: test accuracy, reward, and activation precision (AP). We use test accuracy to measure the predictive performance of the models. As the above section demonstrates, the learned reward model achieves high accuracy in predicting which prototype ranks above another in accordance with human preferences, so we therefore use it as a measure of prototype quality. The final metric, activation precision, is a common metric that has been used in prior work to evaluate the overlap between a prototype's activations and the pixels associated with a given bird (Barnett et al., 2021), which provides another metric of interpretability independent of our method. In our work, we report a modified version of AP introduced in (Bontempelli et al., 2023) to consider the specific value of the activation at each single pixel, not just the overlap alone.

4.1.4. Results

After training ProtoPNet, running the R2 update step, and then performing retraining, we see several trends across multiple base architectures. In Table 1, we report the test accuracy of the different base architectures across stages of R3-ProtoPNet training. Generally, the test accuracy from ProtoPNet temporarily decreases after applying the R2 update, but retraining could effectively recover the predictive loss, in most cases notably improving test accuracy.

In Table 2 and Table 3, we report the average reward and the activation precision metrics. Compared with ProtoPNet, R3-ProtoPNet increases the average reward and activation precision across all prototypes, test images, and base architectures by 27.66% and 18.59%, respectively. Here we note that the average reward and AP serve as complementary interpretability metrics, as in a single stage the reward and AP values across different base architectures could have different patterns - this is because AP only considers the overlap between top 5% activated regions and the bird body, while the reward model/human user takes into account a much larger activated regions with warm colors. However, the increasing patterns for each architecture across different R3 stages are highly consistent, which demonstrate R3's success in aligning with human preference and improving model interpretability.

4.1.5. Discussion

Given the above results, we can observe that although test accuracy experiences a substantial drop during the R2 update, when both reward and AP increase significantly, the model's predictive power is later restored after retraining and in most cases even further improved compared to the original ProtoPNet. On the other hand, the retraining stage doesn't hurt either reward or AP, but instead result in a slight increase of both. This phenomenon suggests that there doesn't exist a long-term trade-off between accuracy and interpretability: this trade-off only temporarily occurs when the spurious feature attributions haven't been removed by our debugging procedure; and once those predictive short-cuts (which tends to have some but limited predictive power) are detected and eliminated, there should be a positive correlation between predictive accuracy and model interpretability. Figure 3 illustrates this empirical trade-off across different ProtoPNet pretraining and R3 debugging stages.

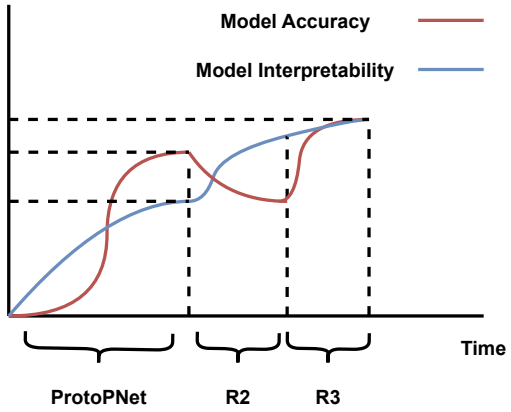

Figure 3. Trade-off curves between model accuracy and model interpretability. The plot is qualitative.

4.2. Car Model Identification

In addition to bird species classification, we also conduct experiments on the Stanford Cars dataset (Krause et al., 2013). This dataset contains 196 different classes with a train/test split of 8144/8041 images. With the implementation remains the same, we found that R3-ProtoPNet outperforms ProtoPNet in a very similar way to the CUB-200-2011 dataset. We include the empirical results of test accuracies and average rewards in Appendix D.

5. Generalizability of the R3 Framework

To test whether the debugging effects of our R3 framework on ProtoPNet could generalize to other models, we incorporate R3 into two other recent models ProtoPFormer (Xue et al., 2022) and ProtoPNet with shallow-deep feature alignment (SDFA) and score aggregation (SA) modules (Huang et al., 2023). With similar experiment setup, we find that although these two models already perform better than ProtoPNet before debugging, our R3 framework is still able to slightly improve them both in terms of accuracy and interpretability. This suggests that our R3 framework could bring forth incremental performance gain to other prototype-based variants. The detailed experiment results can be found in

Table 1. R3 updates tend to increase the test accuracy. Average accuracies and standard deviations are reported across five runs, where m_k is the number of prototypes per class.

BASE (m_k)	PROTOPNET	R2-PROTOPNET	R3-PROTOPNET
VGG-19 (5)	76.33 ± 0.12	62.76 ± 1.18	$\mathbf{77.80} \pm 0.18$
VGG-19 (10)	77.58 ± 0.22	50.41 ± 1.36	$\mathbf{79.60} \pm 0.25$
RESNET-34 (10)	78.73 ± 0.13	58.11 ± 2.71	$\mathbf{80.21} \pm 0.22$
RESNET-50 (10)	78.52 ± 0.17	56.36 ± 2.40	$\mathbf{80.25} \pm 0.22$
DENSENET-121 (10)	79.64 ± 0.23	54.67 ± 2.29	$\mathbf{80.42} \pm 0.26$
DENSENET161 (10)	$\mathbf{79.75} \pm 0.27$	62.75 ± 2.43	79.48 ± 0.36
ENSEMBLE OF ABOVE	82.92 ± 0.09	70.46 ± 0.82	$\mathbf{84.37} \pm 0.20$

Table 2. R3-ProtoPNet outperforms the original ProtoPNet in terms of the image-prototype rewards estimated by our reward model. Values are averaged over the entire test dataset. We divide the R2 update into two columns **Reselected** and **Reweighed** to better show individual effect of each step. We omit the standard deviations as the values are small.

BASE (m_k)	PROTOPNET	RESELECTED	REWEIGHED	R3-PROTOPNET
VGG19 (5)	0.61	0.66	0.70	0.71
VGG19 (10)	0.46	0.55	0.64	0.67
RESNET-34 (10)	0.40	0.47	0.51	0.54
RESNET-50 (10)	0.36	0.45	0.50	0.54
DENSENET-121 (10)	0.48	0.53	0.58	0.58
DENSENET-161 (10)	0.48	0.51	0.57	0.56
AVERAGE	0.47	0.53	0.58	0.60

Table 3. Average Activation Precision (AP) over the test dataset are increased across different stages of R3 updates.

BASE (m_k)	PROTOPNET	RESELECTED	REWEIGHED	R3-PROTOPNET
VGG19 (5)	70.31	79.81	85.64	86.61
VGG19 (10)	63.12	75.95	82.72	81.62
RESNET-34 (10)	85.63	88.81	90.33	92.23
RESNET-50 (10)	71.45	79.29	83.69	83.52
DENSENET-121 (10)	66.22	81.64	86.73	89.38
DENSENET-161 (10)	82.56	85.24	87.55	87.60
AVERAGE	73.22	81.79	86.11	86.83

Appendix E.

6. Limitations and Future Work

While R3 succeeds in bringing forth interpretability and predictive performance gains, there's still room for improvement. For example, the reward model is trained on ratings of individual image-prototype pairs, mostly focusing on overlap and single-image consistency (i.e. whether the prototype simultaneously focuses on multiple body parts in that image), while ignoring cross-image preferences, such as whether the prototype focuses on different parts across images. We also note that R3-ProtoPNet fails to completely eliminate duplicate prototypes, with several high-reward prototypes converge to the same part of the image. To address these issues, it's promising to extend ratings to multiple image-prototype pairs and create more diverse reward models, possibly using them in ensemble. Meanwhile, similar to many other human preference learning scenarios, our ratings to be collected also rely on certain level of subjective individual judgement calls, and this would inevitably lead to noises in the reward labels, which should be further minimized, according to the *predictability, computability, and stability* (PCS) framework for veridical data science (Yu, 2020).

Another limitation with R3-ProtoPNet and other methods that rely on human feedback is that the model itself might be learning features that, while seemingly confusing to a human, are helpful and meaningful for prediction. Barnett et al. (2021) argue that the ProtoPNet can predict with non-obvious features like texture and contrast, which might be

penalized via a learned reward function. An interesting line of future work is to investigate how certain ProtoPNet variants could critique human feedback, and argue against a human-biased reward model.

While this work focuses on improving the performance of ProtoPNet, a major benefit of reward-based finetuning is its flexibility in application. With proper adaptations, we expect our R3 debugging framework could generalize to many other prototype-based or interpretable machine learning models and serve as a useful concept-level debugging tool.

7. Conclusion

In this work, we present the R3 debugging framework, an efficient and generalizable human-in-the-loop approach to improve the class of prototype-based deep learning models. Our work is the first method that uses a learned reward model to quantify the qualitative prototypical visual explanations and use them to improve the model itself. Our experiments show both increased model performance and improved model interpretability. It's demonstrated by our work that the ability of reward learning to quantify qualitative human preferences make reward-based fine-tuning a promising direction for the improvement of interpretable deep models.

Acknowledgements

We would like to acknowledge the partial support from an MSR grant through BAIR at UC Berkeley, NSF grant 2023505 on Collaborative Research: Foundations of Data Science Institute (FODSI), and NSF grant MC2378 to the Institute for Artificial CyberThreat Intelligence and OperatioN (ACTION).

Impact Statement

Our proposed R3 framework is an effective debugging tool for interpretable prototype-based neural networks in the field of Computer Vision, and users could incorporate it into other existing models to improve their performance, thus making more accurate and interpretable decisions. On the other hand, users can use pretrained reward models to evaluate the explanation quality of the models, so they will have a more informed decision on whether to trust the model output.

Furthermore, our framework enables users to train reward models according to their own preferences, which could make the models interpretable in a personalized way.

With all the reasons above, we believe our work will have positive impact on the machine learning and interpretable deep learning communities.

References

Agarwal, A., Tan, Y. S., Ronen, O., Singh, C., and Yu, B. Hierarchical shrinkage: Improving the accuracy and interpretability of tree-based models. In *International Conference on Machine Learning*, pp. 111–135. PMLR, 2022.

Angelov, P. and Soares, E. Towards explainable deep neural networks (xdnn), 2019.

Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., DasSarma, N., et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., Joseph, N., Kadavath, S., Kernion, J., Conerly, T., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Hume, T., Johnston, S., Kravec, S., Lovitt, L., Nanda, N., Olsson, C., Amodei, D., Brown, T., Clark, J., McCandlish, S., Olah, C., Mann, B., and Kaplan, J. Training a helpful and harmless assistant with reinforcement learning from human feedback, 2022.

Barnett, A. J., Schwartz, F. R., Tao, C., Chen, C., Ren, Y., Lo, J. Y., and Rudin, C. Iaia-bl: A case-based interpretable deep learning model for classification of mass lesions in digital mammography, 2021.

Bontempelli, A., Teso, S., Tentori, K., Giunchiglia, F., and Passerini, A. Concept-level debugging of part-prototype networks, 2023.

Bradley, R. A. and Terry, M. E. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Chen, C., Li, O., Tao, D., Barnett, A., Rudin, C., and Su, J. K. This looks like that: deep learning for interpretable image recognition. *Advances in neural information processing systems*, 32, 2019.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Donnelly, J., Barnett, A. J., and Chen, C. Deformable protopnet: An interpretable image classifier using deformable prototypes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10265–10275, June 2022.

Doshi-Velez, F. and Kim, B. Towards a rigorous science of interpretable machine learning, 2017.

Flores-Araiza, D., Lopez-Tiro, F., Villalvazo-Avila, E., El-Beze, J., Hubert, J., Ochoa-Ruiz, G., and Daul, C. Interpretable deep learning classifier by detection of prototypical parts on kidney stones images, 2022.

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

Jeon, H. J., Milli, S., and Dragan, A. Reward-rational (implicit) choice: A unifying formalism for reward learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 4415–4426. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/2f10c1578a0706e06b6d7db6f0b4a6af-Paper.pdf.

Krause, J., Stark, M., Deng, J., and Fei-Fei, L. 3d object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pp. 554–561, 2013. doi: 10.1109/ICCVW.2013.77.

Lage, I., Ross, A. S., Kim, B., Gershman, S. J., and Doshi-Velez, F. Human-in-the-loop interpretability prior, 2018.

Lee, K., Liu, H., Ryu, M., Watkins, O., Du, Y., Boutilier, C., Abbeel, P., Ghavamzadeh, M., and Gu, S. S. Aligning text-to-image models using human feedback. *arXiv preprint arXiv:2302.12192*, 2023.

Lundberg, S. M. and Lee, S.-I. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Ma, C., Zhao, B., Chen, C., and Rudin, C. This looks like those: Illuminating prototypical concepts using multiple visualizations, 2023.

Murdoch, W. J., Liu, P. J., and Yu, B. Beyond word importance: Contextual decomposition to extract interactions from lstms. *arXiv preprint arXiv:1801.05453*, 2018.

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019.

Nauta, M., Jutte, A., Provoost, J., and Seifert, C. This looks like that, because ... explaining prototypes for interpretable image recognition. In *Communications in Computer and Information Science*, pp. 441–456. Springer International Publishing, 2021. doi: 10.1007/978-3-030-93736-2_34. URL https://doi.org/10.1007%2F978-3-030-93736-2_34.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback, 2022.

Ribeiro, M. T., Singh, S., and Guestrin, C. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., and Zhong, C. Interpretable machine learning: Fundamental principles and 10 grand challenges, 2021.

Rymarczyk, D., Struski, L., Tabor, J., and Zieliński, B. Protopshare: Prototypical parts sharing for similarity discovery in interpretable image classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pp. 1420–1430, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467245. URL https://doi.org/10.1145/3447548.3467245.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D. M., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. Learning to summarize from human feedback, 2022.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. J. The caltech-ucsd birds-200-2011 dataset. 2011.

Wang, Y. and Wang, X. Self-interpretable model with transformationequivariant interpretation, 2021.

Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., and Perona, P. Caltech-ucsd birds 200. Technical Report CNS-TR-201, Caltech, 2010. URL /se3/wp-content/uploads/2014/09/WelinderEtal10_CUB-200.pdf,http://www.vision.caltech.edu/visipedia/CUB-200.html.

Xue, M., Huang, Q., Zhang, H., Cheng, L., Song, J., Wu, M., and Song, M. Protopformer: Concentrating on prototypical parts in vision transformers for interpretable image recognition, 2022.

Yu, B. Veridical data science. In *Proceedings of the 13th international conference on web search and data mining*, pp. 4–5, 2020.

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P., and Irving, G. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A. Pairwise Loss Function for the Reward Model

For completeness, here is the explicit formulation of the loss function described in Section 4.2:

$$\mathcal{L}_{\text{reward}} = -\sum_{i \neq i' \text{ or } j \neq j'} \left[\mathbf{1}_{c_{iji'j'}=-1} \log \left(\frac{\exp(r(x_i, h_{ij}))}{\exp(r(x_i, h_{ij})) + \exp(r(x_{i'}, h_{i'j'}))} \right) + \mathbf{1}_{c_{iji'j'}=1} \log \left(\frac{\exp(r(x_{i'}, h_{i'j'}))}{\exp(r(x_i, h_{ij})) + \exp(r(x_{i'}, h_{i'j'}))} \right) \right]$$

where $c_{i,j,i',j'}$ refers to the comparison value associated with the column indexed by i, j, i', j' in the synthetic dataset $\mathcal{D}_{paired}$, which is explained in section 3.3. The architecture of the reward model is detailed in section 4.1.2.

B. Thresholds and the Number of Updated Prototypes

As described in Section 3.4, for each base architecture, the various thresholds for reweighing, reselection, and acceptance. These thresholds were chosen by examining the reward distribution of the base architectures to see if prototypes with low reward cluster around any particular values. Across models, a reweighing threshold of 0.4 or 0.35 sufficed, but further tuning was needed for the reselection and acceptance thresholds. We present the final thresholds used for each R2 step in Table 4.

Table 4. Thresholds used across base architectures during R2 step for the CUB dataset.

BASE (m_k)	RESELECTION THRESHOLD	REWEIGH THRESHOLD	ACCEPTANCE THRESHOLD
VGG-19 (5)	0.35	0.40	0.50
VGG-19 (10)	0.25	0.40	0.43
RESNET-34 (10)	0.22	0.35	0.40
RESNET-50 (10)	0.18	0.35	0.40
DENSENET-121 (10)	0.25	0.35	0.45
DENSENET-161 (10)	0.25	0.35	0.43

Using the reselection thresholds above, we report the total number of updated prototypes for each architecture in Table 5.

Table 5. Total number of prototypes updated across the two R2 steps, divided by the total number of prototypes for that network.

BASE (m_k)	#RESELECTED PROTOTYPES	#REWARD-REWEIGHED PROTOTYPES
VGG-19 (5)	107 / 1000	432 / 1000
VGG-19 (10)	384 / 2000	644 / 2000
RESNET-34 (10)	349 / 2000	702 / 2000
RESNET-50 (10)	365 / 2000	749 / 2000
DENSENET-121 (10)	294 / 2000	662 / 2000
DENSENET-161 (10)	276 / 2000	598 / 2000

C. Sensitivity to Amount of Human Feedback

To evaluate the influence of the amount of human feedback on our R3 framework, we experiment using fewer human ratings to train a reward model and then perform the R3 updates. The results are in Table 6. Although we used 500 ratings to reach the peak performance in the main experiments, it's observed that the R3 framework starts to improve the original ProtoPNet when the number of collected ratings reaches 300.

D. Performance of R3-ProtoPNet on Stanford Cars Dataset

Here we report the test accuracies and average rewards of R3-ProtoPNet on the Stanford Cars dataset in Table 7 and Table 8. Note that although the reward values for this dataset tend to be higher than that of CUB-200-2011, which is due to the

Table 6. Test accuracies of the trained reward models and the debugged R3-ProtoPNets (ensembled) given different amount of human ratings (CUB dataset).

Metric \ #Ratings	**100 ratings**	**200 ratings**	**300 ratings**	**400 ratings**	**500 ratings**
Reward Model Acc.	69.50 ± 0.25	77.34 ± 0.25	83.27 ± 0.27	88.67 ± 0.16	90.09 ± 0.20
R3-ProtoPNet Acc.	77.69 ± 0.30	80.15 ± 0.22	83.06 ± 0.28	84.03 ± 0.21	84.37 ± 0.20

fact that we need to train a new reward model for this new dataset, the general trends are the same. We don't include the activation precision result because a fine-grained segmentation mask for this dataset is not available.

Table 7. R3 updates tend to increase the test accuracy for the Stanford Cars dataset. Average accuracies and standard deviations are reported across five runs, where m_k is the number of prototypes per class.

BASE (m_k)	**PROTOPNET**	**R2-PROTOPNET**	**R3-PROTOPNET**
VGG-19 (5)	85.10 ± 0.15	69.61 ± 1.23	$\mathbf{86.75} \pm 0.18$
VGG-19 (10)	87.25 ± 0.18	66.11 ± 2.10	$\mathbf{88.71} \pm 0.27$
RESNET-34 (10)	85.62 ± 0.08	58.73 ± 2.36	$\mathbf{87.18} \pm 0.14$
RESNET-50 (10)	85.27 ± 0.21	62.66 ± 2.77	$\mathbf{87.25} \pm 0.19$
DENSENET-121 (10)	86.03 ± 0.18	63.49 ± 1.88	$\mathbf{86.59} \pm 0.25$
DENSENET161 (10)	88.19 ± 0.29	60.75 ± 2.23	$\mathbf{89.48} \pm 0.31$
ENSEMBLE OF ABOVE	90.39 ± 0.14	73.42 ± 0.81	$\mathbf{91.57} \pm 0.24$

Table 8. Average rewards during different R3 debugging stages (Stanford Cars dataset).

BASE (m_k)	**PROTOPNET**	**RESELECTED**	**REWEIGHED**	**R3-PROTOPNET**
VGG19 (5)	0.78	0.87	0.91	0.92
VGG19 (10)	0.73	0.82	0.87	0.87
RESNET-34 (10)	0.69	0.75	0.82	0.85
RESNET-50 (10)	0.66	0.74	0.79	0.81
DENSENET-121 (10)	0.75	0.80	0.83	0.86
DENSENET-161 (10)	0.72	0.80	0.82	0.84
AVERAGE	0.72	0.80	0.84	0.86

E. Debugging Effects of R3 on Other ProtoPNet Variants

To test the generalizability of our R3 framework, we first apply our R3 framework to ProtoPFormer (Xue et al., 2022), which is another ProtoPNet extension that uses vision transformer (ViT) backbones. In ProtoPFormer, two types of prototypes are used: the global prototypes are able to provide holistic views of the objects and eliminate confounding effects of the background, while the local prototypes capture the fine-grained visual features that are useful for classification. Empirically we found success applying our R3 framework to update both global and local prototypes. The results are summarized in Table 9.

Huang et al. (2023) propose to improve the stability and consistency of the original ProtoPNet by adding 1) a shallow-deep feature alignment (SDFA) module, which helps preserve the spatial information of deep feature maps by incorporating spatial information from shallow layers into deep layers, and 2) a score aggregation (SA) module, which improves the model by aggregating activation values only into corresponding categories. We apply our R3 framework to this augmented ProtoPNet, and the results are included in Table 10.

These two experiments show that the R3 debugging procedure could generalize well toward other prototype-based models.

Table 9. Performance report of R3-ProtoPFormer across different stages. We used the best-performing DeiT-S backbone.

METRIC	PROTOPFORMER	RESELECTED	REWEIGHED	R3-PROTOPFORMER
TEST ACCURACY	84.27 ± 0.20	73.35 ± 1.52	71.79 ± 2.13	$\mathbf{85.31} \pm 0.23$
AVERAGE REWARD (GLOBAL)	0.55	0.62	0.66	0.68
AVERAGE REWARD (LOCAL)	0.59	0.62	0.67	0.70
ACTIVATION PRECISION (GLOBAL)	83.48	86.59	87.96	87.43
ACTIVATION PRECISION (LOCAL)	86.63	88.28	89.11	89.36

Table 10. Performance report of debugging the augmented ProtoPNet (with SDFA and SA modules), across different R3 stages. We used the best-performing DenseNet-161 backbone.

METRIC	PROTOPNET WITH SDFA AND SA	RESELECTED	REWEIGHED	RETRAINED
TEST ACCURACY	85.03 ± 0.14	75.30 ± 1.82	72.89 ± 2.05	$\mathbf{85.82} \pm 0.22$
AVERAGE REWARD	0.68	0.70	0.74	0.74
ACTIVATION PRECISION	89.21	90.50	92.39	92.88

F. Prototype Examples

Here we provide some examples of prototypes from ProtoPNet, R2-ProtoPNet, and R3-ProtoPNet. In Figure 4, we visualize the changes of the first 5 prototypes within 4 different classes across different stages, by overlaying each prototype on its closest training image patch. A typical trend we observe is that the R2 update indeed centers otherwise off prototypes on the bird in question, and then the complete R3 update makes those prototypes helpful for prediction.

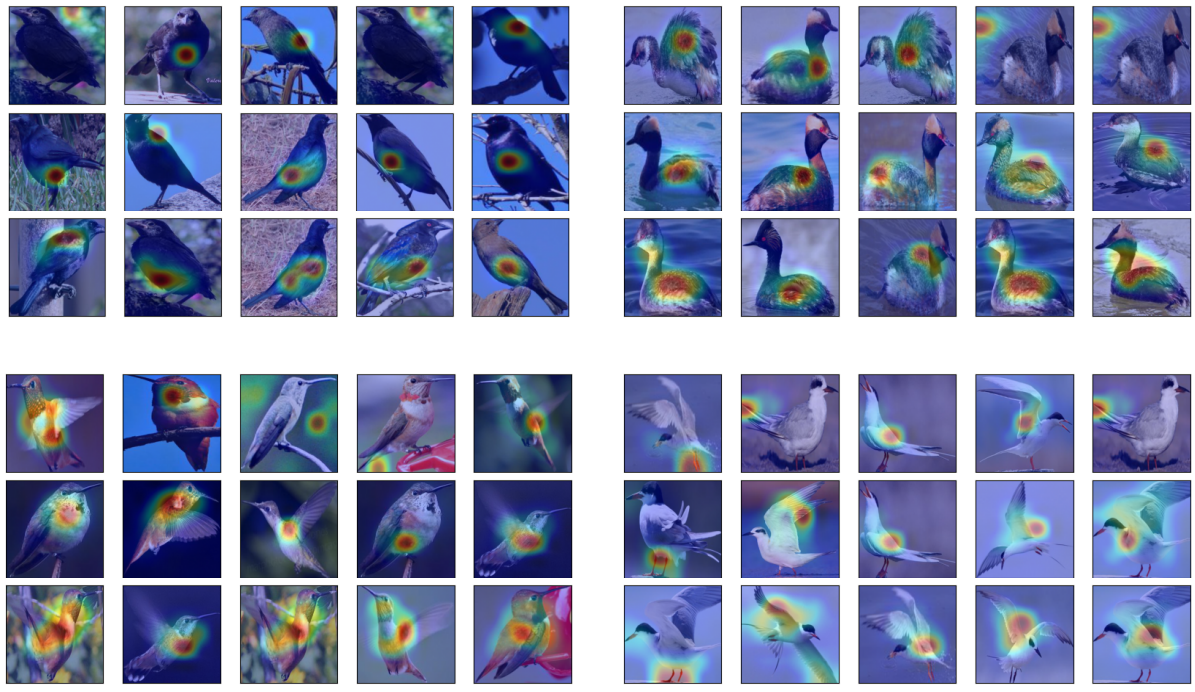

Figure 4. Closest training patches of the five prototypes of ProtoPNet (top row), R2-ProtoPNet (middle row), and R3-ProtoPNet (bottom row) within the same class (5 prototypes per class). Each cluster of 3 rows of images is a seperate class.

In Figure 5, we focus on individual image-prototype pairs. Each column illustrates how the prototype changes during the R2 and R3 update when the image is held fixed. We see that the initially low-quality prototypes are gradually corrected, finally without having much dependence on spurious features like the background.

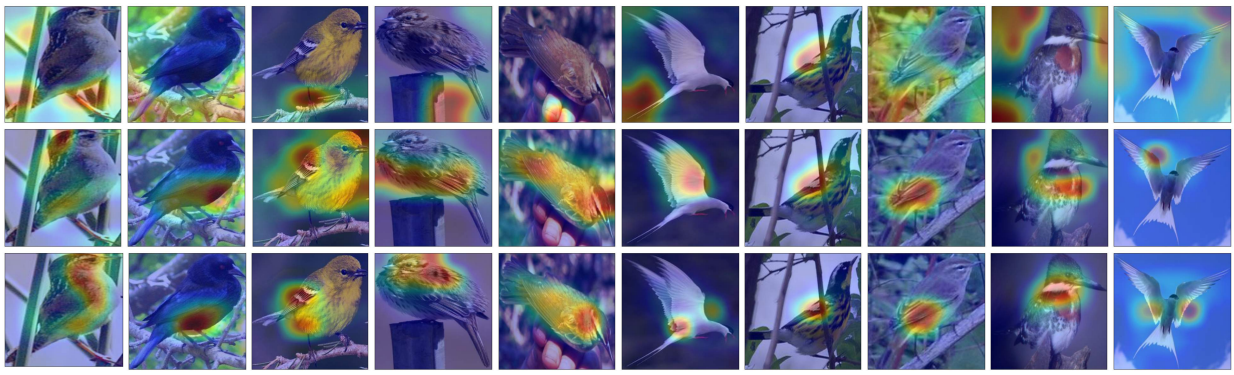

Figure 5. Prototype projections on the same image (each column) from ProtoPNet (top row), R2-ProtoPNet (middle row), and R3-ProtoPNet (bottom row).