

Rashomon Sets for Prototypical-Part Networks: Editing Interpretable Models in Real-Time

Jon Donnelly
Duke University
jon.donnelly@duke.edu

Zhicheng Guo
Duke University
zhicheng.guo@duke.edu

Alina Jade Barnett
Duke University
alina.barnett@duke.edu

Hayden McTavish
Duke University
hayden.mctavish@duke.edu

Chaofan Chen
University of Maine
chaofan.chen@maine.edu

Cynthia Rudin
Duke University
cynthia@cs.duke.edu

Abstract

Interpretability is critical for machine learning models in high-stakes settings because it allows users to verify the model’s reasoning. In computer vision, prototypical part models (ProtoPNets) have become the dominant model type to meet this need. Users can easily identify flaws in ProtoPNets, but fixing problems in a ProtoPNet requires slow, difficult retraining that is not guaranteed to resolve the issue. This problem is called the “interaction bottleneck.” We solve the interaction bottleneck for ProtoPNets by simultaneously finding many equally good ProtoPNets (i.e., a draw from a “Rashomon set”). We show that our framework – called Proto-RSet – quickly produces many accurate, diverse ProtoPNets, allowing users to correct problems in real time while maintaining performance guarantees with respect to the training set. We demonstrate the utility of this method in two settings: 1) removing synthetic bias introduced to a bird-identification model and 2) debugging a skin cancer identification model. This tool empowers non-machine-learning experts, such as clinicians or domain experts, to quickly refine and correct machine learning models without repeated retraining by machine learning experts.

1. Introduction

In high stakes decision domains, there have been increasing calls for interpretable machine learning models [12, 14–16, 35]. This demand poses a challenge for image classification, where deep neural networks offer far superior performance to traditional interpretable models.

Fortunately, case-based deep neural networks have been developed to meet this challenge. These models — in particular, ProtoPNet [7] — follow a simple reasoning process in which input images are compared to a set of learned pro-

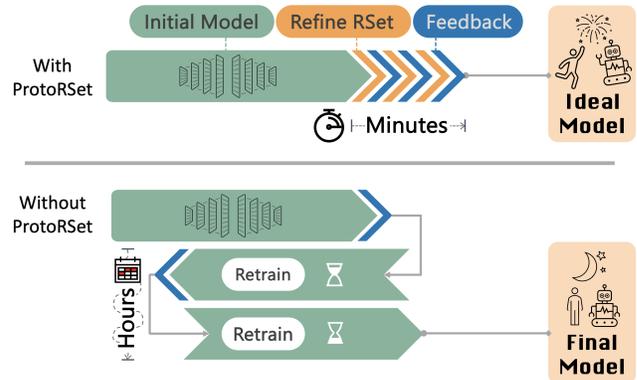


Figure 1. How Proto-RSet addresses the interaction bottleneck. (Bottom) Without Proto-RSet, incorporating user feedback such as “this prototype does not make sense, this would be a better option” requires practitioners to make complicated adjustments to their training regime and train a whole new model. This process can take days, and be prohibitively slow when multiple rounds of feedback are required. (Top) Proto-RSet allows practitioners to incorporate user feedback *in real time* by selecting different candidate models, eliminating the interaction bottleneck. Moreover, Proto-RSet guarantees that user constraints are met, producing their ideal model.

totypes. The similarity of each prototype to a part of the input image is used to form a prediction. This allows users to examine prototypes to check whether the model has learned undesirable concepts and to identify cases that the model considers similar, but that may not be meaningfully related. As such, users can easily identify problems in a trained ProtoPNet. However, they cannot easily *fix* problems in a ProtoPNet because incorporating feedback into a ProtoPNet requires changing model parameters. In the classic paradigm for model troubleshooting, one would retrain the model with added constraints or loss terms to encode user pref-

ferences, but reformulating the problem is time-consuming, and running the algorithm is time-consuming, so repeating this loop even a few times could take days or weeks. This standard troubleshooting paradigm requires alternating input from both domain experts (to provide feedback) and machine learning practitioners (to update the model), which slows down the process. The challenge of troubleshooting models in this classic paradigm has been called the “interaction bottleneck,” and it severely limits the ability of domain experts to create desirable models [36].

In this work, we solve the interaction bottleneck for ProtoPNNets using a different paradigm. We pre-compute many equally-performant ProtoPNNets and allow the domain expert to interact with those. Machine learning experts only need to compute the initial set of models, after which the domain expert can interact with the generated models without interruption. To make this problem computationally feasible, we compute a set of near-optimal models that use *positive reasoning* over a ProtoPNNet with a fixed backbone and set of prototypes. Human interaction with this set is easy and near-instantaneous, allowing the user to choose models that agree with their domain expertise and debug ProtoPNNets easily. The set of near-optimal models for a given problem is called the *Rashomon set*, and thus our approach is called Proto-RSet. While recent work has estimated Rashomon sets for tabular data problems [53, 56], this is the first time it has been attempted for computer vision. Thus, it is the first time users are able to interact nimbly with such a large set of complex models. Figure 1 illustrates the classic model troubleshooting paradigm (lower subfigure), as well as the troubleshooting paradigm from Proto-RSet (upper subfigure), in which we simply provide models that meet a user’s constraints from this set rather than retraining an entire ProtoPNNet; the calculation takes seconds, not days.

Our method for building many good ProtoPNNets is tractable, as are our methods to filter and sample models from this set based on user preferences. We show experimentally that ProtoRSet is feasible to compute in terms of both memory and runtime, and that ProtoRSet allows us to quickly generate many accurate models that are guaranteed to meet user constraints. Finally, we provide two case studies demonstrating the real world utility of Proto-RSet: a user study in which users apply Proto-RSet to rapidly correct biases in a model, and a case study in which we use Proto-RSet to simplify a skin cancer classification model.

2. Related Work

2.1. Interpretable Image Classification

In recent years, a variety of inherently interpretable neural networks for image classification have been developed [3, 23, 44, 55], but case-based interpretable models [7, 28]

have become particularly popular. In particular, ProtoPNNet introduced a popular framework in which images are classified by comparing parts of the image to learned prototypical parts associated with each class. A wide array of extensions have followed the original ProtoPNNet [7]. The majority of these works focus on improving components of the ProtoPNNet itself [10, 30, 32, 33, 37, 38, 47, 48], improving the training regime [34, 39, 52], or applying ProtoPNNets to high stakes tasks [1, 2, 8, 51, 54]. In principle, Proto-RSet can be combined with features from many of these extensions, particularly those that use a linear layer to map from prototype activations to class predictions.

Recently, a line of work integrating human feedback into ProtoPNNets has developed. ProtoPNNetDebug [5] introduced a framework to allow users to inspect a ProtoPNNet to determine if changes might help, then to remove or to require prototypes by re-training the network with loss terms aiming to meet these constraints. The R3 framework [27] used human feedback to develop a reward function, which can be used to form loss terms and guide prototype resampling. In contrast to these approaches, Proto-RSet *guarantees* that accuracy will be maintained, and the constraints given by a user will be met whenever it is possible for such constraints to be met with a well performing model. Our method does not require retraining a neural network.

2.2. The Rashomon Effect

In this work, we leverage the *Rashomon effect* to find many near-optimal ProtoPNNets. The Rashomon effect refers to the observation that, for a given task, there tend to be many disparate, equally good models [6]. This phenomenon presents both challenges and opportunities: the Rashomon effect leads to the related phenomenon of predictive multiplicity [19, 25, 31, 49, 50], wherein equally good models may yield different predictions for any individual, but has also led to theoretical insights into model simplicity [4, 40, 41] and been applied to produce robust measures of variable importance [9, 11, 13, 43]. A more thorough discussion of the implications of the Rashomon effect can be found in [36].

The set of near-optimal models is the *Rashomon set*. The Rashomon set is defined with respect to a specific model class [40]. In recent years, algorithms have been developed to compute or estimate the Rashomon set over decision trees [53], generalized additive models [56], and risk scores [29]. These methods solve deeply challenging computational tasks, but all concern binary classification for tabular data. In this work, we introduce the first method to approximate the Rashomon set for computer vision problems, though our work also applies to other types of signals that are typically analyzed using convolutional neural networks, including, for instance, medical time series (PPG, ECG, EEG).

3. Methods

3.1. Review of ProtoPNets

Before describing Proto-RSet, we first briefly describe ProtoPNets in general, and the training regime followed in computing a reference ProtoPNet. Let $\mathcal{D} := \{\mathbf{X}_i, y_i\}_{i=1}^n$, where $\mathbf{X}_i \in \mathbb{R}^{c \times h \times w}$ is an input image and $y_i \in \{0, 1, \dots, t-1\}$ is the corresponding label. Here, n denotes the number of samples in the dataset, c the number of input channels in each image, h the input image height, w the input image width, and t the number of classes.

A ProtoPNet is an interpretable neural network consisting of three components:

- A backbone $f : \mathbb{R}^{c \times h \times w} \rightarrow \mathbb{R}^{c' \times h' \times w'}$ that extracts a latent representation of each image
- A prototype layer $g : \mathbb{R}^{c' \times h' \times w'} \rightarrow \mathbb{R}^m$ that computes the similarity between each of m prototypes $\mathbf{p}_j \in \mathbb{R}^{c'}$ and the latent representation of a given image, i.e. $g_j(f(\mathbf{X}_i)) = \max_{a,b} \text{sim}(\mathbf{p}_j, f(\mathbf{X}_i)_{:,a,b})$ for some similarity metric sim , where a, b are coordinates along the height and width dimension.¹
- A fully connected layer $h : \mathbb{R}^m \rightarrow \mathbb{R}^t$ that computes an overall classification based on the given prototype similarities. Here, h outputs valid class probabilities (i.e., contains a softmax over class logits).

These layers are optimized for cross entropy and several other loss terms (see [7] for details) using stochastic gradient descent. At inference times, each predicted class probability vector is formed as the composition $\hat{y}_i = h \circ g \circ f(\mathbf{X}_i)$.

3.2. Estimating the Rashomon Set

In general, the Rashomon set of ProtoPNets is defined as:

$$\begin{aligned} \mathcal{R}(\mathcal{D}_{train}; \theta, \ell, \lambda) \\ := \{(\mathbf{w}_f, \mathbf{w}_g, \mathbf{w}_h) \\ : \ell(h(g(f(\cdot; \mathbf{w}_f); \mathbf{w}_g); \mathbf{w}_h), \mathcal{D}_{train}; \lambda) \leq \theta\}, \end{aligned} \quad (1)$$

where ℓ is any regularized loss, λ is the weight of the regularization, θ is the maximum loss allowed in the Rashomon set, and each term \mathbf{w} denotes all parameters associated with the subscripted layer. While \mathcal{R} would be of great practical use, it is intractable to compute: f is typically an extremely complicated, non-convex function, and \mathbf{w}_f and \mathbf{w}_g are extremely high dimensional.

However, if we fix \mathbf{w}_f and \mathbf{w}_g to some reasonable reference values $\bar{\mathbf{w}}_f$ and $\bar{\mathbf{w}}_g$ and instead target

$$\begin{aligned} \bar{\mathcal{R}}(\mathcal{D}_{train}; \theta, \ell, \lambda) := \\ \{\mathbf{w}_h : \ell(h(g(f(\cdot; \bar{\mathbf{w}}_f); \bar{\mathbf{w}}_g); \mathbf{w}_h), \mathcal{D}_{train}; \lambda) \leq \theta\}, \end{aligned} \quad (2)$$

¹In practice, prototypes may consist of multiple spatial components and therefore compare to multiple locations at a time, but for simplicity we only consider prototypes that have spatial size (1×1) in this paper.

we arrive at a much more approachable problem that supports many of the same use cases as \mathcal{R} . In particular, we have reduced (1) to the problem of finding the Rashomon set of multiclass logistic regression models. We describe methods to compute these reference values, with use-cases in the following section, but first let us introduce a method to approximate $\bar{\mathcal{R}}$.

For simplicity of notation, let $\bar{\ell}(\mathbf{w}_h) := \ell(h(g(f(\cdot; \bar{\mathbf{w}}_f); \bar{\mathbf{w}}_g); \mathbf{w}_h), \mathcal{D}_{train}; \lambda)$; that is, the loss of a model where h is parametrized by \mathbf{w}_h and all other components of the ProtoPNet are fixed.

Note that optimizing $\bar{\ell}(\mathbf{w}_h)$ for \mathbf{w}_h is exactly optimizing multiclass logistic regression, which is a convex problem. As such, we can compute the optimal coefficient value \mathbf{w}_h^* with respect to $\bar{\ell}$ using gradient descent. Inspired by [56], we approximate the loss for any coefficient vector \mathbf{w}_h using a second order Taylor expansion:

$$\begin{aligned} \bar{\ell}(\mathbf{w}_h) \\ \approx \bar{\ell}(\mathbf{w}_h^*) + (\nabla \bar{\ell}|_{\mathbf{w}_h^*})^T (\mathbf{w}_h - \mathbf{w}_h^*) \\ + \frac{1}{2} (\mathbf{w}_h - \mathbf{w}_h^*)^T \mathbf{H} (\mathbf{w}_h - \mathbf{w}_h^*) \\ = \bar{\ell}(\mathbf{w}_h^*) + \frac{1}{2} (\mathbf{w}_h - \mathbf{w}_h^*)^T \mathbf{H} (\mathbf{w}_h - \mathbf{w}_h^*), \end{aligned}$$

where \mathbf{H} denotes the Hessian of $\bar{\ell}$ with respect to \mathbf{w}_h . The above equality holds because \mathbf{w}_h^* is the loss minimizer, making $\nabla \bar{\ell}|_{\mathbf{w}_h^*} = \mathbf{0}$. Plugging the above formula for $\bar{\ell}(\mathbf{w}_h)$ into (2), we are interested in finding each \mathbf{w}_h such that

$$\frac{1}{2(\theta - \bar{\ell}(\mathbf{w}_h^*))} (\mathbf{w}_h - \mathbf{w}_h^*)^T \mathbf{H} (\mathbf{w}_h - \mathbf{w}_h^*) \leq 1;$$

this set is an ellipsoid with center \mathbf{w}_h^* and shape matrix $\frac{1}{2(\theta - \bar{\ell}(\mathbf{w}_h^*))} \mathbf{H}$.

This object is convenient to interact with (see Subsection 3.3), but it poses computational challenges. For a standard fully connected layer h , we have $\mathbf{w}_h \in \mathbb{R}^{mt}$ and $\mathbf{H} \in \mathbb{R}^{mt \times mt}$ – this is a prohibitively large matrix, which requires $\mathcal{O}(m^2 t^2)$ floats in memory. The original ProtoPNet [7] used 10 prototypes per class for 200 way classification; with this configuration, a Hessian consisting of 32 bit floats would require 5.12 terabytes to store. In Appendix 1, we describe how models may be sampled from this set using $\mathcal{O}(m^2 + t^2)$ floats in memory.

We can reduce these computational challenges while adhering to a common practitioner preference by requiring *positive reasoning* (i.e., this is class a because it looks like a prototype from class a) rather than *negative reasoning* (i.e., this is class a because it does not look like a prototype from class b). We consider the Rashomon set defined over parameter vectors \mathbf{w}_h that *only allow positive reasoning*, restricting all connections between prototypes and classes other than their assigned class to 0. This allows us to store a Hessian $\mathbf{H} \in \mathbb{R}^{m \times m}$, substantially reducing the memory load

– in the case above, from 5.12 terabytes to 128 megabytes. Appendix 2 provides a derivation of the Hessian for these parameters.

3.3. Interacting With the Rashomon Set

We introduce methods for three common interactions with a Proto-RSet: 1) sampling models from a Proto-RSet; 2) finding a subset of models that *do not* use a given prototype; and 3) finding a subset of models that uses a given prototype with coefficient at least α .

Sampling models from a Proto-RSet: First, we select a random direction vector $\mathbf{d} \in \mathbb{R}^m$ and a random scalar $\kappa \in [0, 1]$. We will produce a parameter vector $\hat{\mathbf{w}}_h := \tau \mathbf{d} + \mathbf{w}_h^*$, where τ is the value such that:

$$\frac{1}{2(\theta - \ell(\mathbf{w}_h^*))} \tau^2 \mathbf{d}^T \mathbf{H} \mathbf{d} = \kappa$$

$$\iff \tau = \sqrt{\frac{2\kappa(\theta - \ell(\mathbf{w}_h^*))}{\mathbf{d}^T \mathbf{H} \mathbf{d}}}$$

The resulting vector $\hat{\mathbf{w}}_h$ is the result of walking κ proportion of the distance from \mathbf{w}_h^* to the border of the Rashomon set along direction \mathbf{d} . This operation allows users to explore individual, equally viable models and get a sense for what a given Proto-RSet will support.

Finding the subset of a Proto-RSet that does not use a given prototype: We say a ProtoPNet does not use a prototype if every weight assigned to that prototype in the last layer is 0. We leverage the fact that Proto-RSet is an ellipsoid to reframe this as the problem of finding the intersection between a hyperplane and an ellipsoid. In particular, to remove prototype j , we define a hyperplane

$$\mathcal{H}_j := \{\mathbf{w}_h : \mathbf{e}_j^T \mathbf{w}_h = 0\},$$

where \mathbf{e}_j is a vector of zeroes, save for a 1 at index j . Thus, to remove prototype j , we compute $\mathcal{R}_{reduced} = \mathcal{H}_j \cap \mathcal{R}$ using the method described in [26]. Note that the intersection between a hyperplane and an ellipsoid is still an ellipsoid, meaning we can still easily perform all operations described in this section on $\mathcal{R}_{reduced}$, including removing multiple prototypes. This operation is useful if, for example, a user wants to remove a prototype that encodes some unwanted bias or confounding.

This procedure provides a natural check for whether or not it is viable to remove a prototype. If \mathcal{H}_j and \mathcal{R} do not intersect, it means that prototype \mathbf{p}_j cannot be removed while remaining in the Rashomon set. This feedback can then be provided to the user, helping them to understand which prototypes are essential for the given problem.

These removals often result in non-trivial changes to the overall reasoning of the model — this is not just setting some coefficients to zero. Figure 2 provides a real example of this phenomena. When prototype 414 is removed,

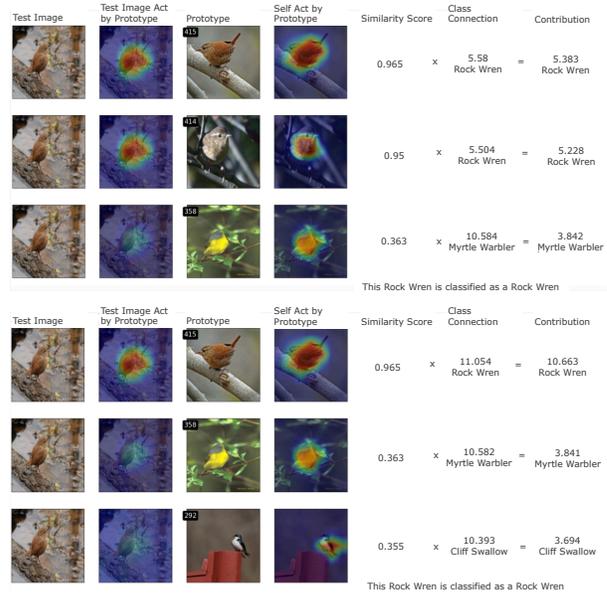


Figure 2. Models produced by ProtoRSet before (top) and after (bottom) a user specifies that prototype 414 must be removed. Proto-RSet guarantees that the bottom model has similar performance to the top, despite following a different reasoning process. If a prototype cannot be removed while maintaining performance, Proto-RSet quickly identifies and reports this.

the weight assigned to prototype 415 is roughly doubled to account for this change in the model, despite being a fairly different prototype.

Finding the subset of a Proto-RSet that uses a prototype with coefficient at least α : Here, we leverage the fact that Proto-RSet is an ellipsoid to reframe this to the problem of finding the intersection between a half-space and an ellipsoid. In particular, to force prototype j to have coefficient $w_j^{(h)} \geq \alpha$, we are interested in the half-space:

$$\mathcal{S}_j(\alpha) := \{\mathbf{w}_h : \mathbf{e}_j^T \mathbf{w}_h \geq \alpha\}.$$

Note that the intersection between an ellipsoid and a half-space is **not** an ellipsoid, meaning we cannot easily perform the operations described in this section on $\mathcal{S}_j(\alpha) \cap \mathcal{R}$. As such, we do not apply these constraints until after prototype removal. We describe a regime to sample models from the Rashomon set after applying multiple halfspace constraints by solving a convex quadratic program in Appendix 3. This operation is useful if, for example, a user finds a prototype that has captured some critical information according to their domain expertise.

3.4. Sampling Additional Candidate Prototypes

When too many constraints are applied, there may be no models in the existing Rashomon set that satisfy all criteria.

Instead of retraining or restarting the model selection process, we can sample additional prototypes in order to expand the Rashomon set. This way, we can retain all existing feedback and continue model refinement.

We generate s new candidate prototypes by randomly selecting patches from the latent representations of the training set images, as we now describe. The backbone $f : \mathbb{R}^{c \times h \times w} \rightarrow \mathbb{R}^{c' \times h' \times w'}$ extracts a latent representation of each image. We randomly select an image i from all training images, and from $f(\mathbf{X}_i)$, we randomly select one c' -length vector from the $h' \times w'$ latent representation. This random selection is repeated s times to provide our s prototype vectors. We then recompute our Proto-RSet with respect to this augmented set of prototypes, and reapply all constraints generated by the user to this point.

4. Experiments

We evaluate Proto-RSet over three fine-grained image classification datasets (CUB-200 [46], Stanford Cars [24], and Stanford Dogs [22]), with ProtoPNets trained on six distinct CNN backbones (VGG-16 and VGG-19 [42], ResNet-34 and ResNet-50 [17], and DenseNet-121 and DenseNet-161 [20]) considered in each case. For each dataset-backbone combination, we applied the Bayesian hyperparameter tuning regime of [52] for 72 GPU hours and used the best model found in terms of validation accuracy after projection as a reference ProtoPNet. For a full description of how these ProtoPNets were trained, see Appendix 4.

We first evaluate the ability of Proto-RSet to quickly produce a set of strong ProtoPNets. Across the eighteen settings described above, we measure the actual runtime required to produce a Proto-RSet given a trained reference ProtoPNet. Appendix 5 describes the hardware used in this experiment. As shown in Figure 3, we find that **a Proto-RSet can be computed in less than 20 minutes across all eighteen settings considered**. This represents a negligible time cost compared to the cost of training one ProtoPNet.

Given that a Proto-RSet can be produced in minutes, we next validate that Proto-RSet quickly produces models that are accurate and meet user constraints. In each of the following sections, we start with a well trained ProtoPNet and iteratively remove up to 100 random prototypes. If we find that no prototype can be removed from the model while remaining in the Rashomon set, we stop this procedure early.

We consider four baselines in the following experiments: naive prototype removal, where all last-layer coefficients for each removed prototype are simply set to 0; naive prototype removal with retraining, where a similar removal procedure is applied and the last-layer of the ProtoPNet is retrained with an ℓ_1 penalty on removed prototype weights; hard removal, where we strictly remove target prototypes and reoptimize all other last layer weights; and ProtoPDebug [5]. Note that we only evaluate ProtoPDebug at 0, 25, 75, and

100 removals due to its long runtime.

Proto-RSet Produces Accurate Models. Figure 4 presents the test accuracy of each model produced in this experiment as a function of the number of prototypes removed. We find that, across all six backbones and all three datasets, **Proto-RSet maintains test accuracy as constraints are added**. In contrast, every method except hard removal shows decreased accuracy as more random prototypes are removed.

Proto-RSet is Fast. Figure 5 presents the time required to remove a prototype using Proto-RSet versus each baseline except naive removal without retraining. We observe that, across all backbones and datasets, Proto-RSet removes prototypes orders of magnitude faster than each baseline method. In fact, Proto-RSet never requires more than a few seconds to produce a model satisfying new user constraints, making prototype removal a viable component of a real-time user interface. Naive removal without retraining tends to be faster, but at the cost of substantial decreases in accuracy.

Proto-RSet Guarantees Constraints are Met. Figure 6 presents the ℓ_1 norm of all coefficients corresponding to removed prototypes. As shown in the figure, **Proto-RSet guarantees that constraints imposed by the user are met**. On the other hand, naive removal of prototypes with retraining does not guarantee that the given constraints are met, with the “removed” prototypes continuing to play a role in the models this method produces. This is because retraining applies these constraints using a loss term, meaning it is not guaranteed that they are met.

4.1. User Study: Removing Synthetic Confounders Quickly

We demonstrate that laypeople can use Proto-RSet to quickly remove confounding from a ProtoPNet. We added synthetic confounding to the training split of the CUB200 dataset and trained a ProtoPNet model using this corrupted data. In Figure 7, we show prototypes that depend on the confounding orange squares added to the Rhinoceros Auklet class. With Proto-RSet, users were able to identify and remove prototypes that depend on these synthetic confounders, producing a corrected model with only 2.1 minutes of computation on average. Compared to the previous state of the art, **Proto-RSet produced models with similar accuracy in roughly one fiftieth of the time**.

To create a confounded model, we modified the CUB200 training dataset such that a model might classify images using the “wrong” features. The CUB200 dataset contains 200 classes, each a different bird species. For the first 100 classes of the training set, we added a colored square to each image where each different color corresponded to a different bird class. The last 100 classes were untouched. We then trained a ProtoPNet with a VGG-19 backbone on

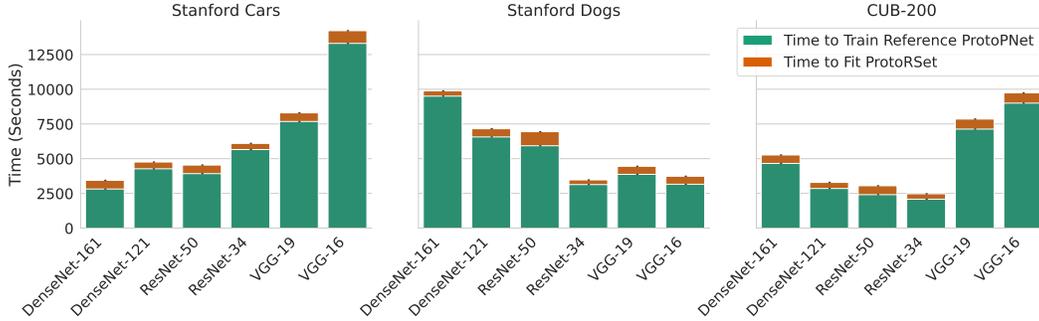


Figure 3. Time to compute Proto-RSet across three datasets and six backbones as a stacked bar plot. The lower bar presents the time required to train the base ProtoPNet, and the top represents the time to compute Proto-RSet given that reference ProtoPNet. We find that Proto-RSet can be computed in less than twenty minutes across a variety of settings.

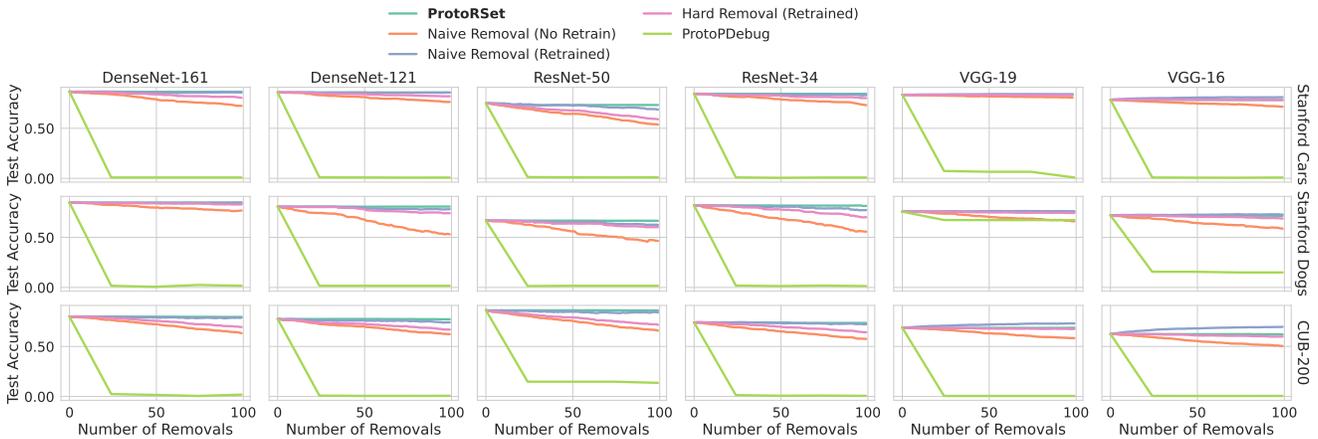


Figure 4. Change in test accuracy as random prototypes are removed. In all cases, we see that removing prototypes using ProtoRSet maintains or slightly improves the accuracy of the original model. Only naive removal with retraining maintains comparable accuracy.

this corrupted dataset. After training, we visually confirmed that this model had learned to reason using confounded prototypes, as shown in Figure 7. For more detail on this procedure, see Appendix 6.

Having trained the confounded model, we recruited 31 participants from the crowd-sourcing platform Prolific to remove the confounded prototypes from the model. Users were instructed to remove each prototype that focused entirely on one of the added color patches as quickly as possible using a simple user interface that allowed them to view and remove prototypes via Proto-RSet, shown in Appendix Figure 11.

We compare to three baseline methods in this setting: ProtoPDebug, and naive prototype removal with and without retraining. We ran each method (Proto-RSet and the three baselines) over the set of prototypes that each user removed. We measured the overall time spent on computation (i.e., not including time spent by the user looking at images). For each baseline method, we collect all prototypes removed by each user and remove them in a single

pass. Note that this is a generous assumption for competitors – whereas the runtime for Proto-RSet includes the time for many refinement calls each time the user updates the model, we only measure the time for a single refinement call for ProtoPDebug and naive retraining. Additionally, we computed the change in validation accuracy after removing each user’s specified prototypes using each method. For ProtoPDebug, we measured the time to remove a set of prototypes as the time taken for a training run to reach its maximum validation accuracy.

Table 1 presents the results of this analysis. Using this interface, we found that users identified and removed an average of 80.8 confounded prototypes and produced a corrected model in an average of 33.6 minutes. Using real user feedback, we find that **Proto-RSet meets user constraints in roughly one fiftieth the time taken by ProtoPDebug**. Moreover, Proto-RSet preserves accuracy more reliably than ProtoPDebug, since ProtoPDebug sometimes produces models with substantially lower accuracy. In Appendix 8, we show that, unlike ProtoPDebug, Proto-RSet

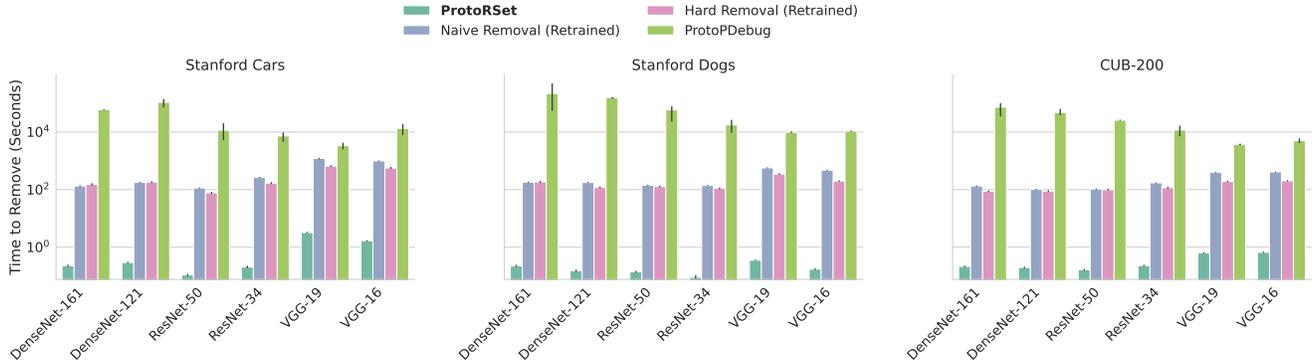


Figure 5. Time in seconds required to remove a single prototype, averaged over 100 iterations of removal. In all cases, ProtoRSet removes prototypes almost instantly. In contrast, removing a prototype then retraining the last layer can take orders of magnitude longer. We exclude naive removal without retraining because it is simply updating a value in an array, and as such is nearly instantaneous.

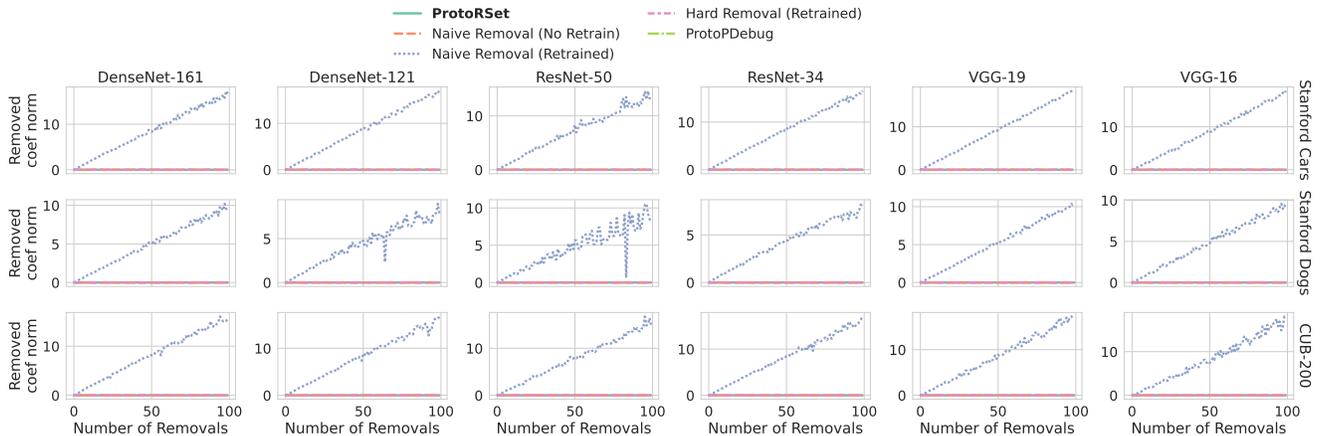


Figure 6. The ℓ_1 norm of all coefficients associated with a “removed” prototype. Here, a larger ℓ_1 norm indicates that the model is not meeting user constraints, since “removed” prototypes are still receiving a large weight in the model. Note that ProtoRSet and naive removal without retraining consistently produce models with an ℓ_1 norm of approximately 0, resulting in overlapping lines. In contrast, naive prototype removal with retraining does not guarantee that prototypes remain removed, resulting in ℓ_1 norms that increase with the number of prototypes we attempt to remove.

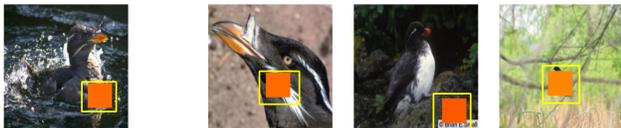


Figure 7. The four most activated images for a confounded prototype from the user study. This prototype has learned to focus on the synthetically added orange patch.

guarantees that user feedback is met.

4.2. Case Study: Refining a Skin Cancer Classification Model

When ProtoPNets are applied to medical tasks, they frequently encounter issues of prototype duplication and ac-

Method	# Removed	Time (Mins)	Δ Acc %
Debug	80.8 ± 44.6	93.7 ± 52.4	$+0.8 \pm 10.4$
Naive	80.8 ± 44.6	0.0 ± 0.0	-6.4 ± 2.9
Naive(T)	80.8 ± 44.6	8.4 ± 0.2	-0.6 ± 0.6
Ours	80.8 ± 44.6	2.1 ± 1.3	-0.5 ± 0.7

Table 1. User study results. Given the prototypes each user requests to remove, we measure mean and standard deviation of the time required to remove each prototype and the change in validation accuracy from the initial ProtoPNet to the resulting model. “Debug” is ProtoPDebug, “Naive (T)” is the naive removal baseline with retraining, “Naive” is naive removal without retraining, and “Ours” is Proto-RSet.

tivation on medically irrelevant areas of the image [1]. We provide an example of Proto-RSet in a realistic, high-

stakes medical setting: skin cancer classification using the HAM10000 dataset [45]. By refining the resulting model using Proto-RSet, **we reduced the total number of prototypes used by the model from 21 to 12 while increasing test accuracy from 70.4% to 71.0%** as shown in Figure 8.

HAM10000 consists of 11,720 skin lesion images, each labeled as one of seven lesion categories that include benign and malignant classifications. We trained a reference ProtoPNet on HAM10000 with a ResNet-34 backbone using the Bayesian hyperparameter tuning from [52] for 48 GPU hours, and selected the best model based on validation accuracy. We used Proto-RSet to refine this model.

Out of 21 unique prototypes, we identified 10 prototypes that either did not focus on the lesion, or duplicated the reasoning of another prototype. We removed 9 of these prototypes. When attempting to remove the final seemingly confounded prototype in Figure 8, our Proto-RSet reported that it was not possible to remove this prototype while remaining in the Rashomon set. We evaluated the accuracy of this claim by manually removing this prototype from the reference ProtoPNet, finding the Proto-RSet with respect to the modified model, and repeating the removals specified above. We found that **ignoring Proto-RSet’s warning decreased test accuracy from 70.4% to 57.9% in the resulting model**. This highlights an advantage of Proto-RSet: when a user tries to impose a constraint that is strongly contradicted by the data, Proto-RSet can directly identify this and prevent unexpected losses in model performance. Proto-RSet produced a sparser, more accurate model even under non-expert refinement.

5. Conclusion

We introduced Proto-RSet, a framework that computes a useful sampling from the Rashomon set of ProtoPNets. We showed that, across multiple datasets and backbone architectures, Proto-RSet consistently produces models that meet all user constraints and maintain accuracy in a matter of seconds. Through a user study, we showed that this enables users to rapidly refine models, and that the accuracy of models from Proto-RSet holds even under feedback from real users. Finally, we demonstrated the real-world utility of Proto-RSet through a case study on skin lesion classification, where we demonstrated both that Proto-RSet can improve accuracy given reasonable user feedback and that Proto-RSet identifies unreasonable changes that *cannot* be made while maintaining accuracy.

It is worth noting that, although we focused on simple ProtoPNets leveraging cosine similarity in this work, Proto-RSet is immediately compatible with many extensions of the original ProtoPNet [7]. Any method that modifies the backbone, the prototype layer, or the loss terms used by ProtoPNet is immediately applicable to Proto-RSet (e.g., [1, 10, 30, 34, 47, 48, 51, 54]), as long as the final prediction

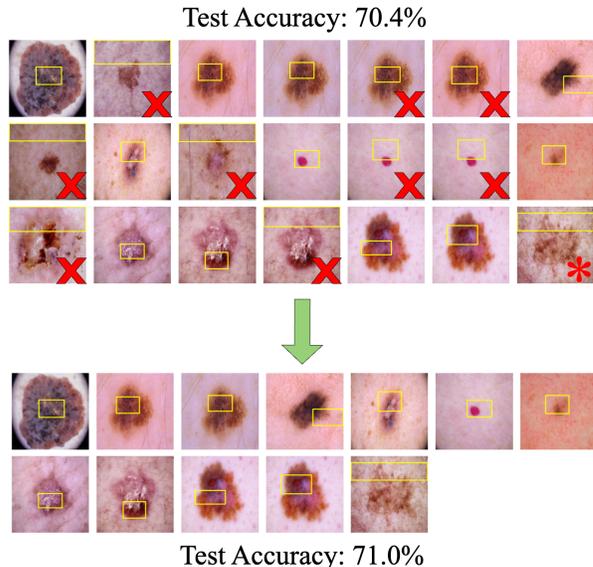


Figure 8. All prototypes from a ProtoPNet trained for skin lesion classification before (top) and after (bottom) refinement using Proto-RSet. Removed prototypes are marked with a red “X”. We attempted to remove the prototype marked with a red “*,” but Proto-RSet correctly identified that this prototype could not be removed without a substantial loss in accuracy.

head is a softmax over a linear layer of prototype similarities. Proto-RSet is also applicable to any future extensions of ProtoPNet that address standing concerns around the interpretability of ProtoPNets [18, 21].

In this work, we set out to solve the deeply challenging problem of interacting with the Rashomon set of ProtoPNets. As such, it is natural that our solution comes with several limitations. First, Proto-RSet is not the entire Rashomon set of ProtoPNets; this means that prior work studying the Rashomon set and its applications may not be immediately applicable to Proto-RSet. Additionally, while Proto-RSet is able to easily require prototypes be used with at least a given weight, doing so makes future operations with Proto-RSet difficult because the resulting object is no longer an ellipsoid.

Nonetheless, Proto-RSet unlocks a new degree of usability for ProtoPNets. Never before has it been possible to debug and refine a ProtoPNet in real time, with guaranteed performance; thanks to Proto-RSet, it is now.

6. Acknowledgements

We acknowledge funding from the National Science Foundation under grants HRD-2222336 and OIA-2218063, and the Department of Energy under grant DE-SC0023194. Additionally, this material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 2139754.

References

- [1] Alina Jade Barnett, Fides Regina Schwartz, Chaofan Tao, Chaofan Chen, Yinhao Ren, Joseph Y Lo, and Cynthia Rudin. A Case-Based Interpretable Deep Learning Model for Classification of Mass Lesions in Digital Mammography. *Nature Machine Intelligence*, 3(12):1061–1070, 2021. 2, 7, 8
- [2] Alina Jade Barnett, Zhicheng Guo, Jin Jing, Wendong Ge, Peter W Kaplan, Wan Yee Kong, Ioannis Karakis, Aline Herlopian, Lakshman Arcot Jayagopal, Olga Taraschenko, et al. Improving Clinician Performance in Classifying EEG Patterns on the Ictal–Interictal Injury Continuum Using Interpretable Machine Learning. *NEJM AI*, 1(6):A10a2300331, 2024. 2
- [3] Moritz Böhle, Mario Fritz, and Bernt Schiele. B-Cos Networks: Alignment Is All We Need for Interpretability. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10329–10338, 2022. 2
- [4] Zachery Boner, Harry Chen, Lesia Semenova, Ronald Parr, and Cynthia Rudin. Using Noise to Infer Aspects of Simplicity Without Learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. 2
- [5] Andrea Bontempelli, Stefano Teso, Katya Tentori, Fausto Giunchiglia, Andrea Passerini, et al. Concept-Level Debugging of Part-Prototype Networks. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 23)*. ICLR 2023, 2023. 2, 5, 7
- [6] Leo Breiman. Statistical Modeling: The Two Cultures (With Comments and a Rejoinder by the Author). *Statistical science*, 16(3):199–231, 2001. 2
- [7] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This Looks Like That: Deep Learning for Interpretable Image Recognition. *Advances in Neural Information Processing Systems*, 32, 2019. 1, 2, 3, 8, 4
- [8] Mohammad Amin Choukali, Mehdi Chehel Amirani, Morteza Valizadeh, Ata Abbasi, and Majid Komeili. Pseudo-Class Part Prototype Networks for Interpretable Breast Cancer Classification. *Scientific Reports*, 14(1):10341, 2024. 2
- [9] Jiayun Dong and Cynthia Rudin. Exploring the Cloud of Variable Importance for the Set of All Good Models. *Nature Machine Intelligence*, 2(12):810–824, 2020. 2
- [10] Jon Donnelly, Alina Jade Barnett, and Chaofan Chen. Deformable Prototnet: An Interpretable Image Classifier Using Deformable Prototypes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10265–10275, 2022. 2, 8, 4
- [11] Jon Donnelly, Srikar Katta, Cynthia Rudin, and Edward Browne. The Rashomon importance distribution: Getting rid of unstable, single model-based variable importance. *Advances in Neural Information Processing Systems*, 36:6267–6279, 2023. 2
- [12] Finale Doshi-Velez and Been Kim. Towards a Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*, 2017. 1
- [13] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All Models Are Wrong, but Many Are Useful: Learning a Variable’s Importance by Studying an Entire Class of Prediction Models Simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019. 2
- [14] US Food and Drug Administration. Machine Learning (AI/ML)-Based Software as a Medical Device (SaMD) Action Plan. *US Food & Drug Administration: Silver Spring, MD, USA*, 2021. 1
- [15] Office for Official Publications of the European Communities. Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act) and Amending Certain Union Legislative Acts. *Proposal for a regulation of the European parliament and of the council*, 2021.
- [16] J Raymond Geis, Adrian P Brady, Carol C Wu, Jack Spencer, Erik Ranschaert, Jacob L Jaremko, Steve G Langer, Andrea Borondy Kitts, Judy Birch, William F Shields, et al. Ethics of Artificial Intelligence in Radiology: Summary of the Joint European and North American Multisociety Statement. *Radiology*, 293(2):436–440, 2019. 1
- [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, 2016. IEEE. 5, 13
- [18] Adrian Hoffmann, Claudio Fanconi, Rahul Rade, and Jonas Kohler. This Looks Like That... Does it? Shortcomings of Latent Space Prototype Interpretability in Deep Networks. *arXiv preprint arXiv:2105.02968*, 2021. 8
- [19] Hsiang Hsu and Flavio Calmon. Rashomon Capacity: A Metric for Predictive Multiplicity in Classification. *Advances in Neural Information Processing Systems*, 35:28988–29000, 2022. 2
- [20] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269. IEEE, 2017. 5, 13
- [21] Qihan Huang, Mengqi Xue, Wenqi Huang, Haofei Zhang, Jie Song, Yongcheng Jing, and Mingli Song. Evaluation and Improvement of Interpretability for Self-Explainable Part-Prototype Networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2011–2020, 2023. 8
- [22] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel Dataset for Fine-Grained Image Categorization: Stanford Dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, 2011. 5, 13
- [23] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept Bottleneck Models. In *International Conference on Machine Learning*, pages 5338–5348. PMLR, 2020. 2
- [24] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013. 5, 13
- [25] Bogdan Kulynych, Hsiang Hsu, Carmela Troncoso, and Flavio P Calmon. Arbitrary Decisions Are a Hidden Cost of

- Differentially Private Training. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pages 1609–1623, 2023. 2
- [26] Alex A Kurzhanskiy and Pravin Varaiya. Ellipsoidal Toolbox (ET). In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 1498–1503. IEEE, 2006. 4
- [27] Aaron Jiaxun Li, Robin Netzorg, Zhihan Cheng, Zhuoqin Zhang, and Bin Yu. Improving Prototypical Visual Explanations With Reward Reweighting, Reselection, and Retraining. In *Forty-first International Conference on Machine Learning*. 2
- [28] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep Learning for Case-Based Reasoning Through Prototypes: A Neural Network That Explains Its Predictions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 2
- [29] Jiachang Liu, Chudi Zhong, Boxuan Li, Margo Seltzer, and Cynthia Rudin. FasterRisk: Fast and Accurate Interpretable Risk Scores. *Advances in Neural Information Processing Systems*, 35:17760–17773, 2022. 2
- [30] Chiyu Ma, Brandon Zhao, Chaofan Chen, and Cynthia Rudin. This Looks Like Those: Illuminating Prototypical Concepts Using Multiple Visualizations. *Advances in Neural Information Processing Systems*, 36, 2024. 2, 8
- [31] Charles Marx, Flavio Calmon, and Berk Ustun. Predictive Multiplicity in Classification. In *International Conference on Machine Learning*, pages 6765–6774. PMLR, 2020. 2
- [32] Meike Nauta, Annemarie Jutte, Jesper Provoost, and Christin Seifert. This Looks Like That, Because... Explaining Prototypes for Interpretable Image Recognition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 441–456. Springer, 2021. 2
- [33] Meike Nauta, Ron Van Bree, and Christin Seifert. Neural Prototype Trees for Interpretable Fine-Grained Image Recognition. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 14933–14943, 2021. 2
- [34] Meike Nauta, Jörg Schlotterer, Maurice Van Keulen, and Christin Seifert. Pip-Net: Patch-Based Intuitive Prototypes for Interpretable Image Classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2744–2753, 2023. 2, 8
- [35] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable Machine Learning: Fundamental Principles and 10 Grand Challenges. *Statistic Surveys*, 16:1–85, 2022. 1
- [36] Cynthia Rudin, Chudi Zhong, Lesia Semenova, Margo Seltzer, Ronald Parr, Jiachang Liu, Srikanth Katta, Jon Donnelly, Harry Chen, and Zachery Boner. Amazing Things Come From Having Many Good Models. In *Proceedings of the International Conference on Machine Learning*, 2024. 2
- [37] Dawid Rymarczyk, Łukasz Struski, Jacek Tabor, and Bartosz Zieliński. ProtoPShare: Prototypical Parts Sharing for Similarity Discovery in Interpretable Image Classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1420–1430, 2021. 2
- [38] Dawid Rymarczyk, Łukasz Struski, Michał Górszczak, Koryna Lewandowska, Jacek Tabor, and Bartosz Zieliński. Interpretable Image Classification With Differentiable Prototypes Assignment. In *European Conference on Computer Vision*, pages 351–368. Springer, 2022. 2
- [39] Dawid Rymarczyk, Joost van de Weijer, Bartosz Zieliński, and Bartłomiej Twardowski. ICICLE: Interpretable Class Incremental Continual Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1887–1898, 2023. 2
- [40] Lesia Semenova, Cynthia Rudin, and Ronald Parr. On the Existence of Simpler Machine Learning Models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1827–1858, 2022. 2
- [41] Lesia Semenova, Harry Chen, Ronald Parr, and Cynthia Rudin. A Path to Simpler Models Starts With Noise. *Advances in Neural Information Processing Systems*, 36, 2024. 2
- [42] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015. 5, 13
- [43] Gavin Smith, Roberto Mansilla, and James Goulding. Model Class Reliance for Random Forests. *Advances in Neural Information Processing Systems*, 33:22305–22315, 2020. 2
- [44] Mohammad Reza Taesiri, Giang Nguyen, and Anh Nguyen. Visual Correspondence-Based Explanations Improve AI Robustness and Human-AI Team Accuracy. *Advances in Neural Information Processing Systems*, 35:34287–34301, 2022. 2
- [45] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The HAM10000 Dataset, a Large Collection of Multi-Source Dermatoscopic Images of Common Pigmented Skin Lesions. *Scientific data*, 5(1):1–9, 2018. 8
- [46] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 5, 13
- [47] Chong Wang, Yuyuan Liu, Yuanhong Chen, Fengbei Liu, Yu Tian, Davis McCarthy, Helen Frazer, and Gustavo Carneiro. Learning Support and Trivial Prototypes for Interpretable Image Classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2062–2072, 2023. 2, 8
- [48] Jiaqi Wang, Huaifeng Liu, Xinyue Wang, and Liping Jing. Interpretable Image Recognition by Constructing Transparent Embedding Space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 895–904, 2021. 2, 8
- [49] Jamelle Watson-Daniels, Solon Barocas, Jake M Hofman, and Alexandra Chouldechova. Multi-Target Multiplicity: Flexibility and Fairness in Target Specification Under Resource Constraints. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pages 297–311, 2023. 2
- [50] Jamelle Watson-Daniels, David C Parkes, and Berk Ustun. Predictive Multiplicity in Probabilistic Classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 10306–10314, 2023. 2

- [51] Yuanyuan Wei, Roger Tam, and Xiaoying Tang. MProtoNet: A Case-Based Interpretable Model for Brain Tumor Classification With 3D Multi-Parametric Magnetic Resonance Imaging. In *Medical Imaging with Deep Learning*, pages 1798–1812. PMLR, 2024. [2](#), [8](#)
- [52] Frank Willard, Luke Moffett, Emmanuel Mokel, Jon Donnelly, Stark Guo, Julia Yang, Giyoung Kim, Alina Jade Barnett, and Cynthia Rudin. This Looks Better Than That: Better Interpretable Models With ProtoPNext. *arXiv preprint arXiv:2406.14675*, 2024. [2](#), [5](#), [8](#), [4](#), [9](#), [13](#)
- [53] Rui Xin, Chudi Zhong, Zhi Chen, Takuya Takagi, Margo Seltzer, and Cynthia Rudin. Exploring the Whole Rashomon Set of Sparse Decision Trees. *Advances in Neural Information Processing Systems*, 35:14071–14084, 2022. [2](#)
- [54] Julia Yang, Alina Jade Barnett, Jon Donnelly, Satvik Kishore, Jerry Fang, Fides Regina Schwartz, Chaofan Chen, Joseph Y Lo, and Cynthia Rudin. FPN-Iaia-BI: A Multi-Scale Interpretable Deep Learning Model for Classification of Mass Margins in Digital Mammography. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5003–5009, 2024. [2](#), [8](#)
- [55] Weiqiu You, Helen Qu, Marco Gatti, Bhuvnesh Jain, and Eric Wong. Sum-of-Parts Models: Faithful Attributions for Groups of Features. *arXiv preprint arXiv:2310.16316*, 2023. [2](#)
- [56] Chudi Zhong, Zhi Chen, Jiachang Liu, Margo Seltzer, and Cynthia Rudin. Exploring and Interacting With the Set of Good Sparse Generalized Additive Models. *Advances in Neural Information Processing Systems*, 36, 2024. [2](#), [3](#)

Rashomon Sets for Prototypical-Part Networks: Editing Interpretable Models in Real-Time

Supplementary Material

1. Sampling Unrestricted Last Layers from the Rashomon Set Without Explicitly Computing the Hessian

Let $\mathbf{z} \in \mathbb{R}^m$ denote a vector of prototype similarities, such that $h(\mathbf{z}) \in \mathbb{R}^t$ is the vector of predicted class probabilities for input \mathbf{z} . The Hessian matrix of h with respect to \mathbf{z} for a standard, multi-class logistic regression problem is given (written in terms of block matrices) as:

$$\begin{aligned} \mathbf{H} &= - \begin{bmatrix} h_1(\mathbf{z})(1-h_1(\mathbf{z}))\mathbf{z}\mathbf{z}^T & -h_1(\mathbf{z})h_2(\mathbf{z})\mathbf{z}\mathbf{z}^T & \dots & -h_1(\mathbf{z})h_t(\mathbf{z})\mathbf{z}\mathbf{z}^T \\ -h_2(\mathbf{z})h_1(\mathbf{z})\mathbf{z}\mathbf{z}^T & h_2(\mathbf{z})(1-h_2(\mathbf{z}))\mathbf{z}\mathbf{z}^T & \dots & -h_2(\mathbf{z})h_t(\mathbf{z})\mathbf{z}\mathbf{z}^T \\ \vdots & \vdots & \ddots & \vdots \\ -h_t(\mathbf{z})h_1(\mathbf{z})\mathbf{z}\mathbf{z}^T & -h_t(\mathbf{z})h_2(\mathbf{z})\mathbf{z}\mathbf{z}^T & \dots & h_t(\mathbf{z})(1-h_t(\mathbf{z}))\mathbf{z}\mathbf{z}^T \end{bmatrix} \\ &= (\mathbf{\Lambda} - h(\mathbf{z})h(\mathbf{z})^T) \otimes \mathbf{z}\mathbf{z}^T, \end{aligned}$$

where

$$\mathbf{\Lambda}_{ij} := \mathbf{1}[i = j]h_i(\mathbf{z}).$$

In order to sample a member of a Rashomon set that falls along direction $\mathbf{d} \in \mathbb{R}^{mt}$, we need to compute a value τ such that

$$\tau = \sqrt{\frac{2\kappa(\theta - \ell(\mathbf{w}_h^*))}{\mathbf{d}^T \mathbf{H} \mathbf{d}}},$$

with each term defined as in Section 3.3 of the main paper. The key computational constraint here is the operation $\mathbf{d}^T \mathbf{H} \mathbf{d}$, which involves the large Hessian matrix $\mathbf{H} \in \mathbb{R}^{mt \times mt}$. However, the quantity $\mathbf{d}^T \mathbf{H} \mathbf{d}$ can be computed without explicitly storing \mathbf{H} .

Let $\text{mat} : \mathbb{R}^{mt} \rightarrow \mathbb{R}^{m \times t}$ denote an operation that reshapes a vector into a matrix, and let $\text{vec} : \mathbb{R}^{m \times t} \rightarrow \mathbb{R}^{mt}$ denote the inverse operator, which reshapes a matrix into a vector such that $\text{vec}(\text{mat}(\mathbf{d})) = \mathbf{d}$. We can then leverage the property of the Kronecker product that $(\mathbf{A} \otimes \mathbf{B})\text{vec}(\mathbf{C}) = \text{vec}(\mathbf{B}\mathbf{C}\mathbf{A}^T)$ to compute:

$$\begin{aligned} \mathbf{d}^T \mathbf{H} \mathbf{d} &= \mathbf{d}^T ((\mathbf{\Lambda} - h(\mathbf{z})h(\mathbf{z})^T) \otimes \mathbf{z}\mathbf{z}^T) \mathbf{d} \\ &= \mathbf{d}^T ((\mathbf{\Lambda} - h(\mathbf{z})h(\mathbf{z})^T) \otimes \mathbf{z}\mathbf{z}^T) \text{vec}(\text{mat}(\mathbf{d})) \\ &= \mathbf{d}^T \text{vec} \left(\underbrace{\mathbf{z}\mathbf{z}^T}_{m \times m} \underbrace{\text{mat}(\mathbf{d})}_{m \times t} \underbrace{(\mathbf{\Lambda} - h(\mathbf{z})h(\mathbf{z})^T)^T}_{t \times t} \right) \end{aligned}$$

As highlighted by the shape annotations, this operation can be computed by storing matrices of no larger than $\max(m^2, t^2, mt)$.

2. Positive Connections Only

For both memory efficiency and conceptual simplicity, instead of learning the Rashomon set over all last layers, we might want to consider one with some parameter tying. In particular, we allow prototypes to connect only with their own class. We begin with notation: let $S_c := \{i : \psi(i) = c\}$ where c is a class and $\psi(i)$ is a function that returns the class associated with prototype i . The constrained last layer forms predictions as:

$$h_c^{lin}(\mathbf{x}) := \sum_{i \in S_c} w_i x_i \quad (3)$$

$$h(\mathbf{x}) = \text{softmax}(h^{lin}(\mathbf{x})) \quad (4)$$

The first partial derivative of cross entropy w.r.t. a parameter w_i is:

$$\begin{aligned} \frac{\partial}{\partial w_i} CE(f(\mathbf{x}), \mathbf{y}) &= \frac{\partial}{\partial w_i} \left[\sum_{k=1}^C y_k \left(h_k^{lin}(\mathbf{x}) - \ln \left(1 + \sum_{k=1}^C \exp(h_k^{lin}(\mathbf{x})) \right) \right) \right] \\ &= y_{\psi(i)} \frac{\partial}{\partial \omega_i^{(+)}} h_{\psi(i)}^{lin}(\mathbf{x}) - \frac{\exp(h_{\psi(i)}^{lin}(\mathbf{x}))}{1 + \sum_{k=1}^C \exp(h_k^{lin}(\mathbf{x}))} \frac{\partial}{\partial \omega_i^{(+)}} h_{\psi(i)}^{lin}(\mathbf{x}) \\ &= y_{\psi(i)} x_i - \frac{\exp(h_{\psi(i)}^{lin}(\mathbf{x}))}{1 + \sum_{k=1}^C \exp(h_k^{lin}(\mathbf{x}))} x_i \\ &= (y_{\psi(i)} - h_{\psi(i)}(\mathbf{x})) x_i \end{aligned}$$

For the second derivative, we have:

$$\begin{aligned} \frac{\partial}{\partial w_j} \frac{\partial}{\partial w_i} CE(f(\mathbf{x}), \mathbf{y}) &= \frac{\partial}{\partial w_j} (y_{\psi(i)} - h_{\psi(i)}(\mathbf{x})) x_i \\ &= -x_i \frac{\partial}{\partial w_j} h_{\psi(i)}(\mathbf{x}) \\ &= -x_i \sum_{k=1}^C \frac{\partial h_{\psi(i)}}{\partial h_k^{lin}} \frac{\partial h_k^{lin}}{\partial w_j} \\ &= -x_i \frac{\partial h_{\psi(i)}}{\partial h_{\psi(j)}^{lin}} \frac{\partial h_{\psi(j)}^{lin}}{\partial w_j} \\ &= -x_i x_j h_{\psi(i)}(\mathbf{x}) (\delta_{\psi(i)\psi(j)} - h_{\psi(j)}(\mathbf{x})) \end{aligned}$$

Packaged together, we then have

$$\begin{aligned} \mathbf{H} &:= \begin{bmatrix} -x_1^2 h_{\psi(1)}(\mathbf{x}) (\delta_{\psi(1)\psi(1)} - h_{\psi(1)}(\mathbf{x})) & -x_2 x_1 h_{\psi(1)}(\mathbf{x}) (\delta_{\psi(2)\psi(1)} - h_{\psi(2)}(\mathbf{x})) & \dots \\ -x_1 x_2 h_{\psi(2)}(\mathbf{x}) (\delta_{\psi(1)\psi(2)} - h_{\psi(1)}(\mathbf{x})) & -x_2^2 h_{\psi(2)}(\mathbf{x}) (\delta_{\psi(2)\psi(2)} - h_{\psi(2)}(\mathbf{x})) & \dots \\ \dots & \dots & \dots \\ -x_1 x_d h_{\psi(d)}(\mathbf{x}) (\delta_{\psi(1)\psi(d)} - h_{\psi(1)}(\mathbf{x})) & -x_2 x_d h_{\psi(d)}(\mathbf{x}) (\delta_{\psi(2)\psi(d)} - h_{\psi(2)}(\mathbf{x})) & \dots \end{bmatrix} \\ &= \begin{bmatrix} h_{\psi(1)}(\mathbf{x}) (\delta_{\psi(1)\psi(1)} - h_{\psi(1)}(\mathbf{x})) & h_{\psi(1)}(\mathbf{x}) (\delta_{\psi(2)\psi(1)} - h_{\psi(2)}(\mathbf{x})) & \dots \\ h_{\psi(2)}(\mathbf{x}) (\delta_{\psi(1)\psi(2)} - h_{\psi(1)}(\mathbf{x})) & h_{\psi(2)}(\mathbf{x}) (\delta_{\psi(2)\psi(2)} - h_{\psi(2)}(\mathbf{x})) & \dots \\ \dots & \dots & \dots \\ h_{\psi(d)}(\mathbf{x}) (\delta_{\psi(1)\psi(d)} - h_{\psi(1)}(\mathbf{x})) & h_{\psi(d)}(\mathbf{x}) (\delta_{\psi(2)\psi(d)} - h_{\psi(2)}(\mathbf{x})) & \dots \end{bmatrix} \odot -\mathbf{xx}^T \end{aligned}$$

where \odot denotes a Hadamard product.

3. Sampling From a Rashomon Set After Requiring Prototypes

Let J denote a set of prototype indices we wish to require, such that we constrain $[\text{coef}(\mathbf{p}_j) \geq \alpha] \forall j \in J$ where $\text{coef}(\mathbf{p}_j)$ denotes the last layer coefficient associated with prototype \mathbf{p}_j and $\alpha \in \mathbb{R}$ is the minimum acceptable coefficient. Given this information, there are a number of ways to formulate requiring prototypes as a convex optimization problem. One such form, which we use, is to optimize the following:

$$\min_{\mathbf{w}_h} \frac{1}{2(\theta - \bar{\ell}(\mathbf{w}_h^*))} (\mathbf{w}_h - \mathbf{w}_h^*)^T \mathbf{H} (\mathbf{w}_h - \mathbf{w}_h^*) \quad \text{s.t. } [\mathbf{e}_j^T \mathbf{w}_h \geq \alpha] \forall j \in J$$

We then check whether the result for $(\mathbf{w}_h - \mathbf{w}_h^*)^T \mathbf{H} (\mathbf{w}_h - \mathbf{w}_h^*)$ satisfies the constraint of being < 1 . If so, we've found weights within our Rashomon set approximation. If not, we've proved no such solution exists.

4. Training Details for Reference ProtoPNets

Proto-RSet is created by first training a reference ProtoPNet, then calculating a subset of the Rashomon set based on that reference model. Each reference ProtoPNet was trained using the Bayesian hyperparameter optimization framework described in [52]. We used cosine similarity for prototype comparisons. We trained each backbone using four training phases, as described in [7]: warm up, in which only the prototype layer and add-on layers (additional convolutional layers appended to the end of the backbone of a ProtoPNet) are optimized; joint, in which all backbone, prototype layer, and add-on layer parameters are optimized; project, in which prototypes are set to be exactly equal to their nearest neighbors in the latent space; and last-layer only, in which only the final linear layer is optimized.

We trained each model to minimize the following loss term:

$$\ell_{total} = CE + \lambda_{clst}\ell_{clst} + \lambda_{sep}\ell_{sep} + \lambda_{ortho}\ell_{ortho},$$

where each λ term is a hyperparameter coefficient, CE is the standard cross entropy loss, ℓ_{clst} and ℓ_{sep} are the cluster and separation loss terms from [7] adapted for cosine similarity, and ℓ_{ortho} is orthogonality loss as defined in [10]. In particular, our adaptations of cluster and separation loss are defined as:

$$\ell_{clst} := \frac{1}{n} \sum_{i=1}^n \max_{j \in \{1, \dots, m\} : \text{class}(\mathbf{p}_j) = y_i} g_j(f(\mathbf{X}_i))$$

$$\ell_{sep} := \frac{1}{n} \sum_{i=1}^n \max_{j \in \{1, \dots, m\} : \text{class}(\mathbf{p}_j) \neq y_i} g_j(f(\mathbf{X}_i)),$$

where $\text{class}(\mathbf{p}_j)$ is a function that returns the class with which prototype \mathbf{p}_j is associated. During last-layer only optimization, we additionally minimize the ℓ_1 norm of the final linear layer weights with a hyperparameter coefficient λ_{ℓ_1} . For each of our experiments, we performed Bayesian optimization with the prior distributions specified in Table 2 and used the model with the highest validation accuracy following projection as our reference ProtoPNet. We deviated from this selection criterion only for the user study; for that experiment, we selected a model with a large gap between train and validation accuracy (indicating overfitting to the induced confounding), and visually confirmed the use of confounded prototypes.

Hyperparameter Name	Distribution	Description
pre_project_phase_len	Integer Uniform; Min=3, Max=15	The number of warm and joint optimization epochs to run before the first prototype projection is performed
post_project_phases	Fixed value; 10	Number of times to perform projection and the subsequent last layer only and joint optimization epochs
phase_multiplier	Fixed value; 1	Amount to multiply each number of epochs by, away from a default of 10 epochs per training phase
lr_multiplier	Normal; Mean=1.0, Std=0.4	Amount to multiply all learning rates by, relative to the reference values in [52]
joint_lr_step_size	Integer Uniform; Min=2, Max=10	The number of training epochs to complete before each learning rate step, in which each learning rate is multiplied by 0.1
num_addon_layers	Integer Uniform; Min=0, Max=2	The number of additional convolutional layers to add between the backbone and the prototype layer
latent_dim_multiplier_exp	Integer Uniform; Min=-4, Max=1	If num_addon_layers is not 0, dimensionality of the embedding space will be multiplied by $2^{\text{latent_dim_multiplier}}$ relative to the original embedding dimension of the backbone
num_prototypes_per_class	Integer Uniform; Min=8, Max=16	The number of prototypes to assign to each class
cluster_coef	Normal; Mean=-0.8, Std=0.5	The value of λ_{clst}
separation_coef	Normal; Mean=0.08, Std=0.1	The value of λ_{sep}
l1_coef	Log Uniform; Min=0.00001, Max=0.001	The value of λ_{ℓ_1}
orthogonality_coef	Log Uniform; Min=0.00001, Max=0.001	The value of λ_{ortho}

Table 2. Prior distributions over hyperparameters for the Bayesian sweep used to train all reference ProtoPNets except for the one in the user study.

5. Hardware Details

All of our experiments were run on a large institutional compute cluster. We trained each reference ProtoPNet using a single NVIDIA RTX A5000 GPU with CUDA version 12.4, and ran other experiments using a single NVIDIA RTX A6000 GPU with CUDA version 12.4.

6. Confounding Details

To run our user study, we trained a reference ProtoPNet over a version of CUB200 in which a confounding patch has been added to each training image. For each training image coming from one of the first 100 classes, we add a color patch with width and height equal to $1/5$ those of the image to a random location in the image. The color of each patch is determined by the class of the training image; class 0 is set to the initial value for the HSV color map in matplotlib, class 1 to the color $1/100^{\text{th}}$ further along this color map, and so on. A sample from each class with this confounding is shown in Figure 9. Images from the validation and test splits of the dataset were not altered.



Figure 9. One example image from each class in CUB200, with confounding patches added as in the user study. The first 100 classes receive random confounding patches, and the second 100 are unaltered.

7. User Study Details

In this section, we describe our user study in further detail.

7.1. Setup

We followed the procedure described in Appendix 6 to create a confounded version of the CUB200 training set, and fit a ProtoPNet with a VGG-19 backbone over this confounded set. We created this model using a Bayesian hyperparameter sweep, but rather than selecting the model with the highest validation accuracy, we selected the model with the largest gap between its train and validation accuracy, as this indicates overfitting to the confounding patches added to the training set. The hyperparameters used for the selected model are shown in Table 3. Each ProtoPDebug model trained in this experiment used these hyperparameters, and the prescribed “forbid loss” from [5] with a coefficient of 100 as in [5].

We visually examined the prototypes learned by this model to identify the minimal set of “gold standard” prototypes we expected users to remove. We found that, of 1509 prototypes, 27 were clearly confounded, with bounding boxes focused entirely on a confounding patch and global analyses illustrating that the three most similar training patches to each prototype were also confounding patches. Figure 10 shows all prototypes used by this model, as well as the 27 “gold standard” confounded prototypes we identified. Figure 11 shows a screenshot of the interface for our user study.

7.2. Recruitment and Task Description

Users were recruited from the crowd sourcing platform Prolific, and were tasked with removing prototypes that clearly focused on confounding patches. The complete, anonymized informed consent text shown to users – which provides instructions – was as follows:

This research study is being conducted by **REDACTED**. This research examines whether a novel tool can be used to remove obvious errors in an AI model. You will be asked to use the tool to remove obvious errors, a task that will take approximately 30 minutes and no longer than an hour. Your participation in this research study is voluntary.

You may withdraw at any time, but you will only be paid if you remove at least 10% of color-patch prototypes and do not remove more than 2 non-color-patch prototypes. After completing the survey, you will be paid at a rate of \$15/hour of work through the Prolific platform.

In accordance with Prolific policies, we may reject your work if the task was not completed correctly, or the instructions were not followed. A bonus payment of \$10 will be offered if all errors are identified and corrected within 30 minutes.

There are no anticipated risks or benefits to participants for participating in this study. Your participation is anonymous as we will not collect any information that the researchers could identify you with.

If you have any questions about this study, please contact **REDACTED** and include the term “Prolific Participant Question” in your email subject line. For questions about your rights as a participant contact **REDACTED**.

If you consent to participate in this study please click the “>>” below to begin the survey.

A total of 51 Prolific users completed our survey. Of these, 20 submissions were rejected for either 1) failing to identify at least 10% of the gold standard confounded prototypes (i.e., identified 2 or fewer) or 2) removing more than 2 prototypes drawn from images with no confounding patch. A total of 4 users qualified for and received the \$10 bonus payment.

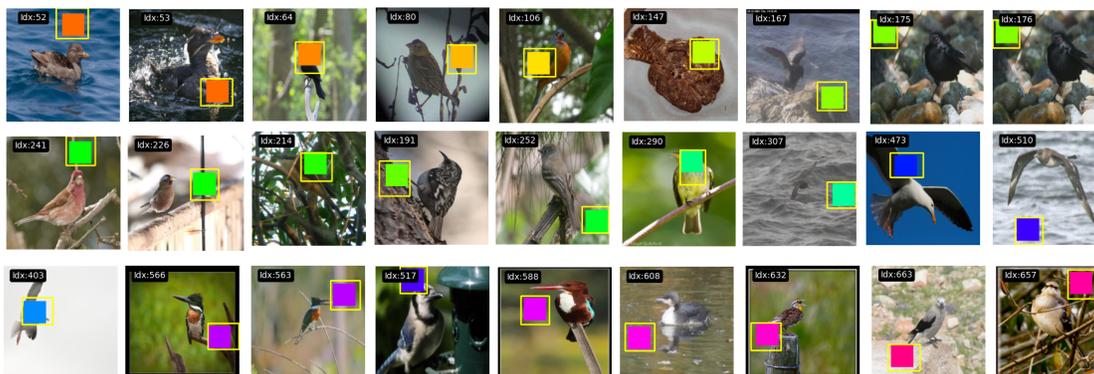
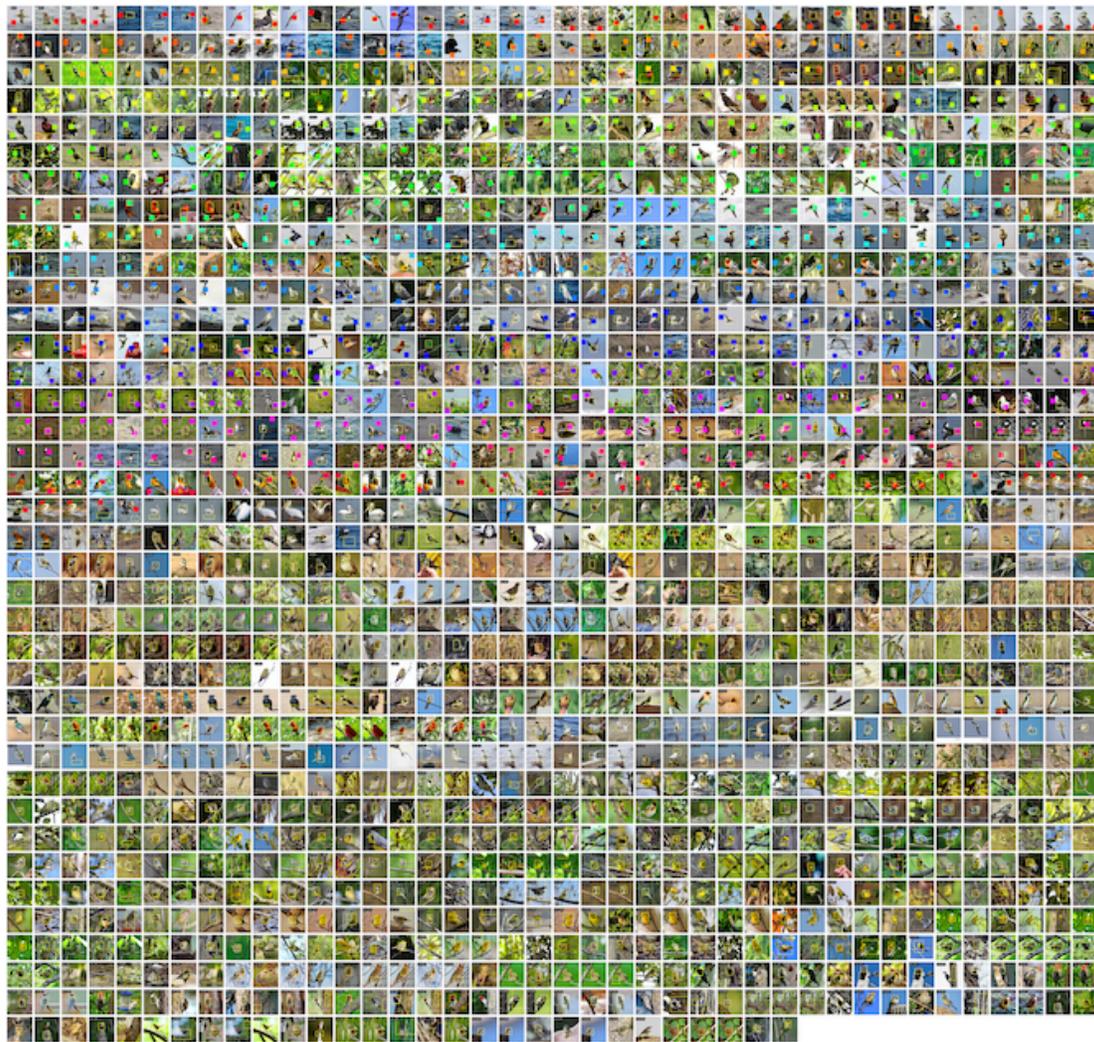


Figure 10. **(Top)** All 1509 prototypes used by the confounded model from the user study. **(Bottom)** The 27 clearly confounded prototypes we identified as the “gold standard” for users to remove.

Hyperparameter Name	Value	Description
pre_project_phase_len	11	The number of warm and joint optimization epochs to run before the first prototype projection is performed
post_project_phases	10	Number of times to perform projection and the subsequent last layer only and joint optimization epochs
phase_multiplier	1	Amount to multiply each number of epochs by, away from a default of 10 epochs per training phase
lr_multiplier	0.89	Amount to multiply all learning rates by, relative to the reference values in [52]
joint_lr_step_size	8	The number of training epochs to complete before each learning rate step, in which each learning rate is multiplied by 0.1
num_addon_layers	1	The number of additional convolutional layers to add between the backbone and the prototype layer
latent_dim_multiplier_exp	-4	If num_addon_layers is not 0, dimensionality of the embedding space will be multiplied by $2^{\text{latent_dim_multiplier}}$ relative to the original embedding dimension of the backbone
num_prototypes_per_class	14	The number of prototypes to assign to each class
cluster_coef	-1.2	The value of λ_{clst}
separation_coef	0.03	The value of λ_{sep}
l1_coef	0.00001	The value of λ_{ℓ_1}
orthogonality_coef	0.0004	The value of λ_{ortho}

Table 3. Hyperparameter values used to construct the confounded ProtoPNet for the user study.

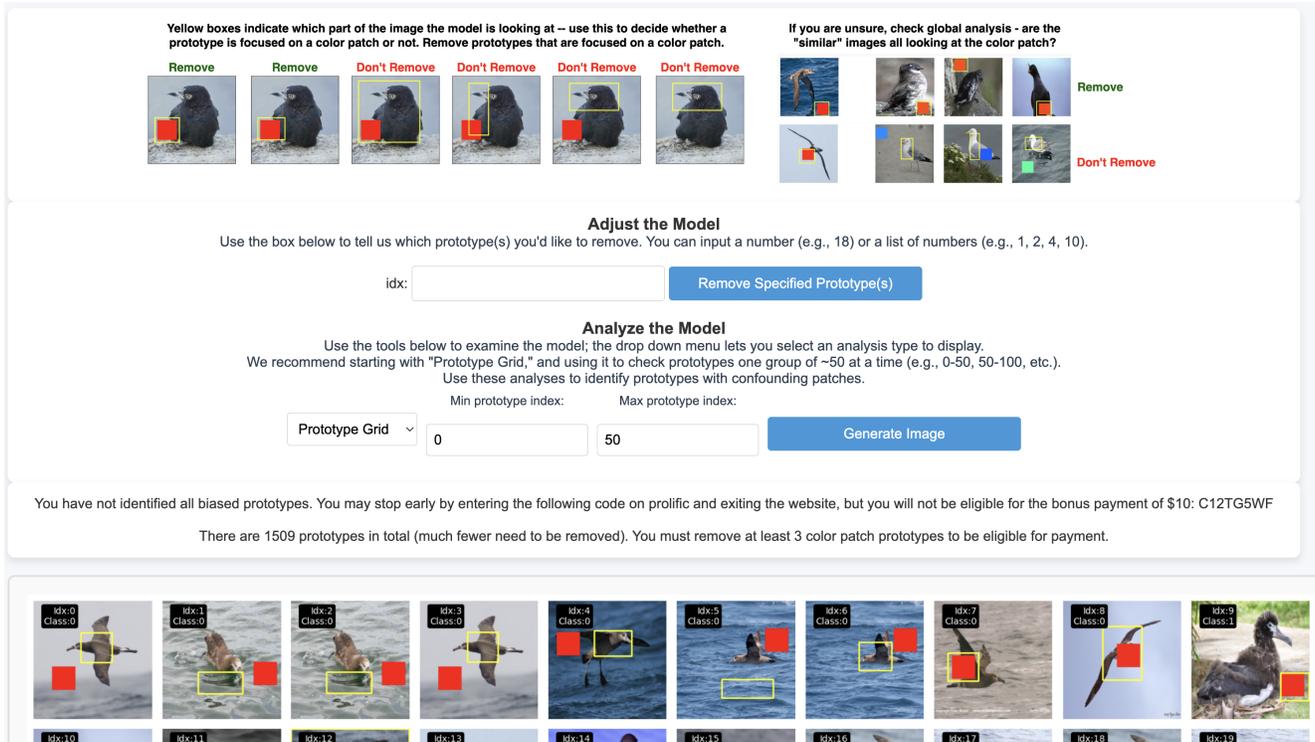


Figure 11. A screenshot of the interface for our user study. Users were tasked with identifying and removing confounded prototypes.

8. Proto-RSet Meets Feedback Better than ProtoPDebug

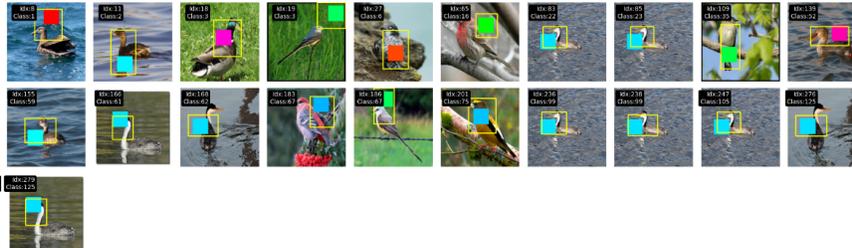
Here, we use results from the user study to highlight a key advantage in Proto-RSet over ProtoPDebug: Proto-RSet guarantees that user constraints are met when possible. We selected a random user who successfully identified all “gold standard” confounded prototypes (see Figure 10). With this user’s feedback, we created two models: one using ProtoPDebug and one from Proto-RSet. We examined all prototypes used by each model. In both models, we identified every prototype for which the majority of the prototype’s bounding box focused on a confounding color patch. Figure 12 shows every confounded prototype from the original reference model, the model produced by ProtoPDebug, and the model produced by Proto-RSet. Note that, for the original reference model, we visualize every prototype marked as confounded by the user plus those we identified as confounded. We mark “false positive” user feedback (prototypes that were removed, but do not focus on confounding patches) with a red “X,” and do not count them toward the total number of confounded prototypes for the original model.

We found that a substantial portion of the prototypes used by ProtoPDebug – 6.3% of all prototypes – still used confounded prototypes despite user feedback. In contrast, only 0.5% of the prototypes used by Proto-RSet focused on confounding patches. Moreover, the 7 confounded prototypes that remain after applying Proto-RSet would not be present if the user had identified them; this is not guaranteed for the 21 confounded prototypes of ProtoPDebug.

Original Model: 12.1% confounded (182 / 1509)



ProtoPDebug: 6.3% confounded (21 / 332)



Proto-RSet: 0.5% confounded (7 / 1327)



Figure 12. All confounded prototypes from each of a reference ProtoPNet, a model produced by ProtoPDebug, and a model produced by Proto-RSet. A prototype is considered confounded if the majority of the bounding box is covered by a confounding patch. Proto-RSet and ProtoPDebug were shown the feedback obtained from the same random user. A substantially larger proportion of the prototypes from ProtoPDebug are confounded than the prototypes from Proto-RSet.

9. Detailed Parameter Values for Proto-RSet

Here, we briefly describe the parameters used when computing the Proto-RSet’s from each experimental section. Each Rashomon set was defined with respect to an ℓ_2 regularized cross entropy loss $\bar{\ell}(\mathbf{w}_h) = CE(\mathbf{w}_h) + \lambda\|\mathbf{w}_h\|_2$, with $\lambda = 0.0001$. In all of our experiments except for the user study, we used a Rashomon parameter of $\theta = 1.1\bar{\ell}(\mathbf{w}_h^*)$; for the user study, we set $\theta = 1.2\bar{\ell}(\mathbf{w}_h^*)$ to account for the fact that the training set was confounded, meaning training loss was a less accurate indicator of model performance. We estimated the empirical loss minimizer \mathbf{w}_h^* using stochastic gradient descent with a learning rate of 1.0 for a maximum of 5,000 epochs. We stopped this optimization early if loss decreased by no more than 10^{-7} between epochs.

10. Evaluating Proto-RSet’s Ability to Require Prototypes

In this section, we evaluate the ability of Proto-RSet to require prototypes using the method described in the main paper. We match the experimental setup from the main prototype removal experiments; namely, we evaluate Proto-RSet over three fine-grained image classification datasets (CUB-200 [46], Stanford Cars [24], and Stanford Dogs [22]), with ProtoPNETs trained on six distinct CNN backbones (VGG-16 and VGG-19 [42], ResNet-34 and ResNet-50 [17], and DenseNet-121 and DenseNet-161 [20]) considered in each case. For each dataset-backbone combination, we applied the Bayesian hyperparameter tuning regime of [52] for 72 GPU hours and used the best model found in terms of validation accuracy after projection as a reference ProtoPNET. For a full description of how these ProtoPNETs were trained, see Appendix 4.

In each of the following experiments, we start with a well-trained ProtoPNET and iteratively require that up to 100 random prototypes have coefficient greater than τ , where τ is the mean value of the non-zero entries in the reference ProtoPNET’s final linear layer. If we find that no prototype can be required from the model while remaining in the Rashomon set, we stop this procedure early. This occurred n times.

We consider two baselines for comparison in each of the following experiments: naive prototype requirement, in which the correct-class last-layer coefficient for each required prototype is set to τ and no further adjustments are made to the model, and naive prototype requirement with retraining, in which a similar requirement procedure is applied, but the last-layer of the ProtoPNET is retrained (with all other parameters held constant) after prototype requirement. In the second baseline, we apply an ℓ_1 reward to coefficients corresponding to required prototypes to prevent the model from forgetting them and train the last layer until convergence, or for up to 5,000 epochs – whichever comes first. By ℓ_1 reward, we mean that this quantity is *subtracted* from the overall loss value, so that a larger value for these entries decreases loss.

Proto-RSet Produces Accurate Models. Figure 13 presents the test accuracy of each model produced in this experiment as a function of the number of prototypes required. We find that, across all six backbones and all three datasets, **Proto-RSet maintains test accuracy as constraints are added.** This stands in stark contrast to direct requirement of undesired prototypes, which dramatically reduces performance in all cases. The test accuracy of models produced by Proto-RSet is maintained **in all cases.**

Proto-RSet is Fast. Figure 14 presents the time necessary to require a prototype using Proto-RSet versus naive prototype requirement with retraining. We observe that, across all backbones and datasets, Proto-RSet requires prototypes orders of magnitude faster than retraining a model. In contrast to prototype removal, we observe that the time necessary to require prototypes is heavily dependent on the reference ProtoPNET. Naive requirement without retraining tends to be faster, but at the cost of substantial decreases in accuracy.

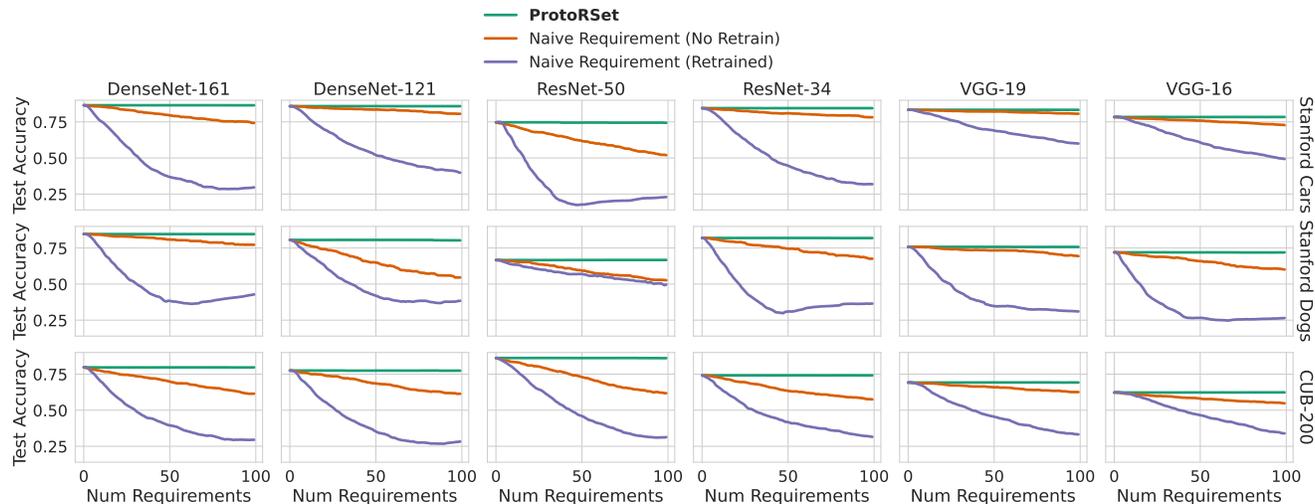


Figure 13. Change in test accuracy as random prototypes are required. In all cases, we see that requiring prototypes using ProtoRSet maintains or slightly improves the accuracy of the original model. Naively requiring prototypes either with or without retraining, on the other hand, dramatically reduces model performance.

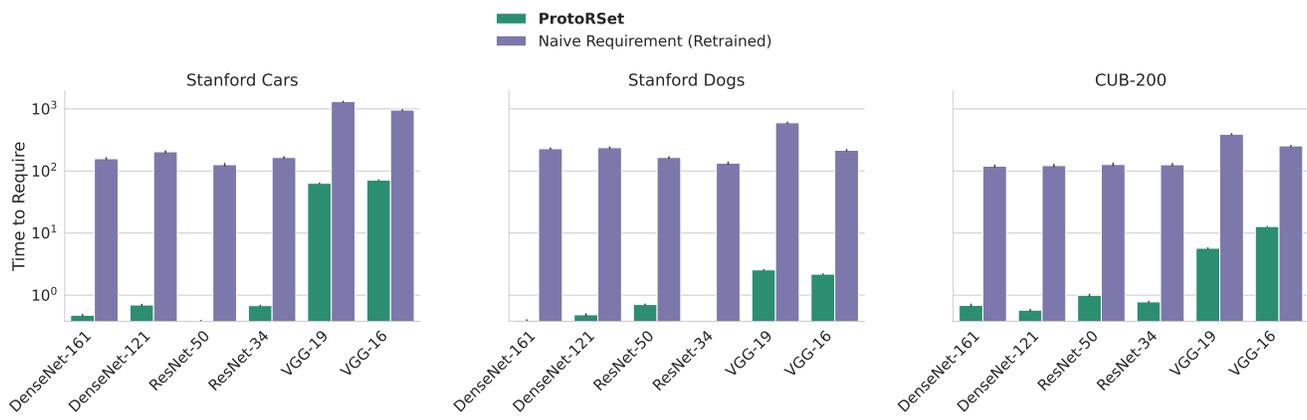


Figure 14. Time in seconds to produce a model meeting requirement constraints, averaged over 100 iterations of requirement. In all cases, ProtoRSet meets the prototype requirement constraint faster than the naive method (requiring a prototype then retraining the last layer). We exclude naive requirement without retraining from the chart because it is simply updating a value in an array, and as such is nearly instantaneous.

11. Evaluating the Sampling of Additional Prototypes

In this section, we evaluate whether the prototype sampling mechanism described in subsection 3.4 allows users to impose additional constraints by revisiting the skin cancer classification case study from subsection 4.2. In subsection 4.2, we attempted to remove 10 of the original 21 prototypes used by the model, and found that we were unable to remove the one of these prototypes without sacrificing accuracy.

We sampled 25 additional prototypes using the mechanism described in subsection 3.4 using the same reference ProtoP-Net, and analyzed the resulting model for any additional background or duplicate prototypes. We found that 9 out of the 25 additional prototypes focused primarily on the background. We removed all 10 of the original target prototypes and these 9 new ones, resulting in a model with 27 prototypes in total, as shown in Figure 15. **By sampling additional prototypes, Proto-RSet was able to meet user constraints that were not possible given the original set of prototypes.** This model achieved identical test accuracy to the original model. Recalling that accuracy dropped substantially when removing the 10 target prototypes without sampling alternatives, this demonstrates that sampling more prototypes increases the flexibility of Proto-RSet.

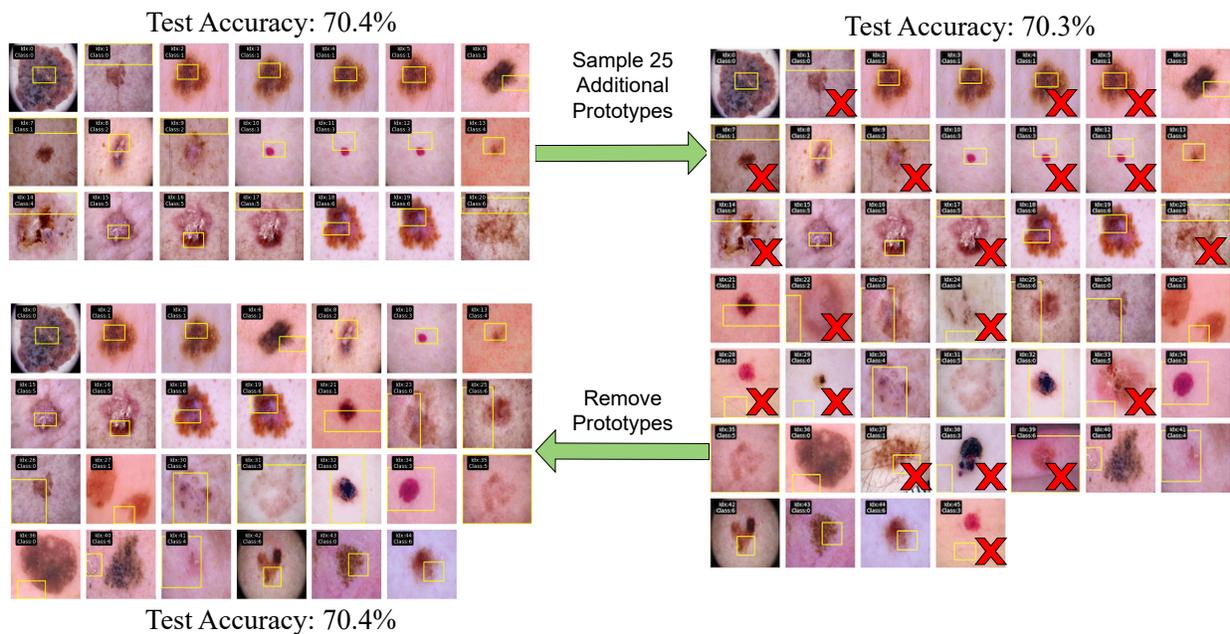


Figure 15. The process followed to remove all desired prototypes from a model for skin cancer classification. Starting from a reference ProtoPNet with 21 prototypes (Top Left), we first sampled 25 additional prototypes to produce a set of 46 candidate prototypes (Right). We removed 19 prototypes from this set of candidates, each of which is annotated with a red "X". Removing these prototypes using Proto-RSet resulted in a model with 27 non-confounded prototypes that matched the test accuracy of the original model.

12. Additional Visualizations of Model Refinement

In this section, we provide additional visualizations of model reasoning before and after user constraints are added. We consider the ResNet-50 based ProtoPNet trained for CUB-200 that we used for our experiments in the main body. The changes in prototype class-connection weight show that substantial changes to model reasoning are achieved when editing models using Proto-RSet.

Figure 16 presents the reasoning process of the model in classifying a Least Auklet before and after an arbitrary prototype is removed. In this case, we see that Proto-RSet substantially adjusted the weight assigned to prototypes of the same class as the removed prototype, and slightly adjusted the coefficient on prototypes from the class with the second highest logit – in this case, the Parakeet Auklet class. Notably, the weight assigned to prototype 12 increases from 5.014 to 9.029.

Figure 17 presents the reasoning process of the model in classifying a Black-footed Albatross before and after we require a large coefficient be assigned to an arbitrary prototype. We see that Proto-RSet substantially decreased the weight assigned to all other prototypes of the same class as the upweighted prototype, and moderately changed the coefficient on prototypes from the class with the second highest logit – in this case, the Sooty Albatross class. In particular, the weight assigned to prototype 6 was increased from 6.934 to 7.11, and the weight on prototype 5 was decreased from 6.87 to 6.786.

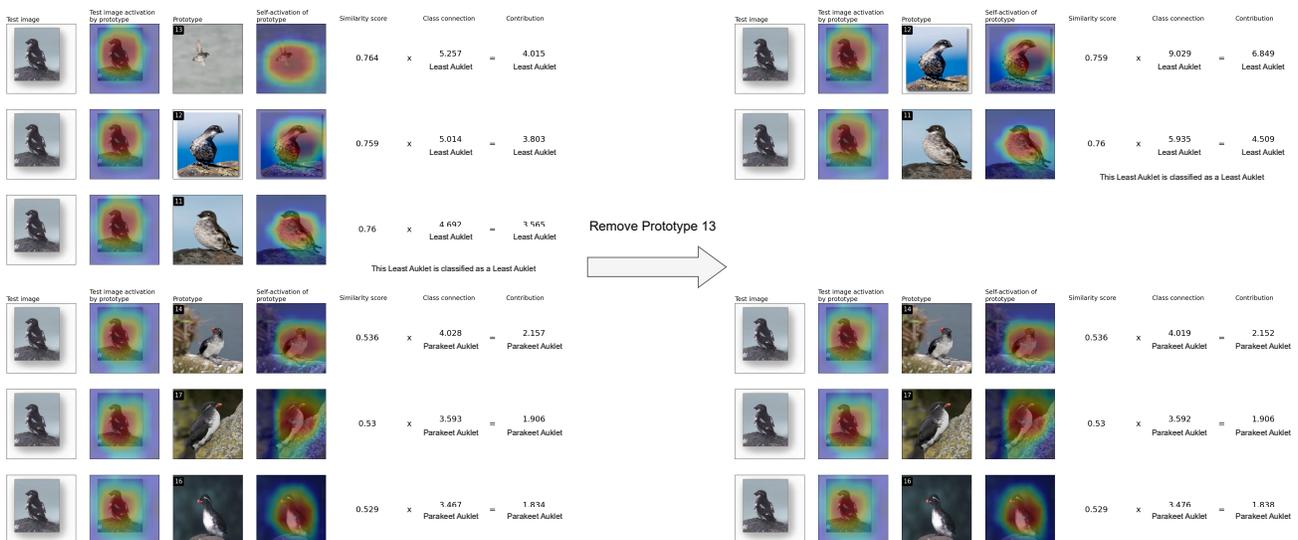


Figure 16. Reasoning process for a ProtoPNet classifying a Least Auklet before (Left) and after (Right) prototype 13 is removed using Proto-RSet. In each column, we present the reasoning for the predicted class (Top) and the class with the second highest logit (Bottom).

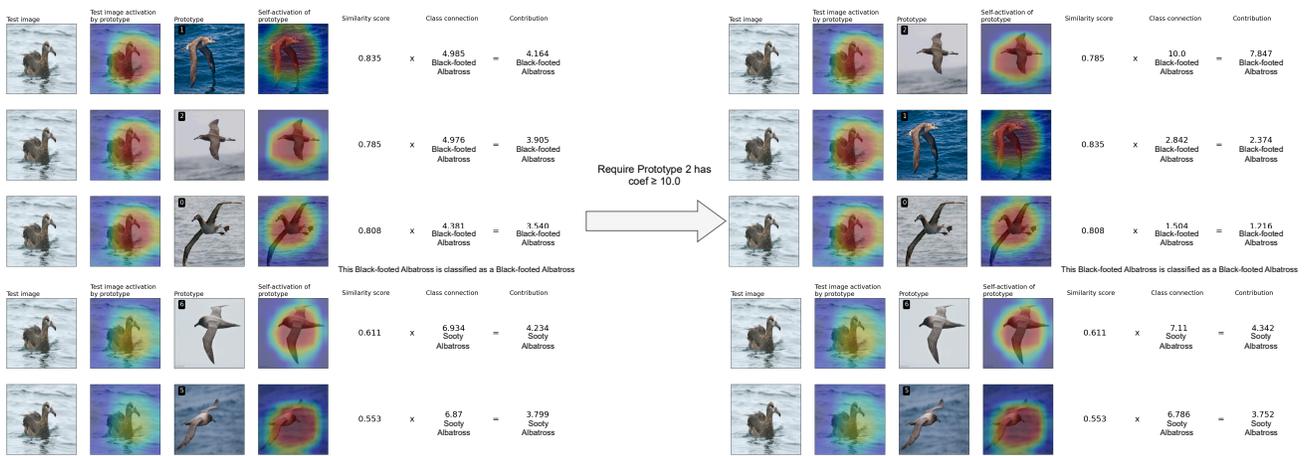


Figure 17. Reasoning process for a ProtoPNet classifying a Black-footed Albatross before (Left) and after (Right) the model is constrained such that prototype 2 has a coefficient of at least 10 using Proto-RSet. In each column, we present the reasoning for the predicted class (Top) and the class with the second highest logit (Bottom).