AutoProtoNet: Interpretability for Prototypical Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

In meta-learning approaches, it is difficult for a practitioner to make sense of what kind of representations the model employs. Without this ability, it can be difficult to both understand what the model knows as well as to make meaningful corrections. To address these challenges, we introduce AutoProtoNet, which builds interpretability into Prototypical Networks by training an embedding space suitable for reconstructing inputs, while remaining convenient for few-shot learning. We demonstrate how points in this embedding space can be visualized and used to understand class representations. We also devise a prototype refinement method, which allows a human to debug inadequate classification parameters. We use this debugging technique on a custom classification task and find that it leads to accuracy improvements on a validation set consisting of in-the-wild images. We advocate for interpretability in meta-learning approaches and show that there are interactive ways for a human to enhance meta-learning algorithms.

1 Introduction

It is expensive and time-consuming to collect data to train current state-of-the-art image classification systems [13]. When a classification algorithm is deployed, new classes or labels cannot be easily added without incurring new costs related to re-training the model [1][2]. Meta-learning approaches for few-shot learning solve both these problems by training networks that learn quickly from little data with computationally inexpensive fine-tuning [23][20][15]. Despite these methods performing well on benchmark few-shot image classification tasks, these methods are not interpretable; a human may have no way of knowing why a certain classification decision was made. Additionally, the lack of interpretability limits any kind of debugging of network representations. In this work, we take a step toward the development of a meta-learning algorithm which can learn in a few-shot setting, can handle new classes at test time, is interpretable enough for a human to understand how the model makes decisions, and which can be debugged in a simple way.

We revisit Prototypical Networks (ProtoNets) [20] as the focus of our study. ProtoNets are based on a simple idea: there exists an embedding space where images cluster around a single "prototype" for each class. Given the simplicity of this few-shot learning approach, it makes sense to ask: what does a prototype look like? And, have we learned an adequate prototype representation?

The outcomes of our study can be summarized as follows:

- We introduce AutoProtoNet, which merges ideas from autoencoders and Prototypical Networks, to perform few-shot image classification and prototype reconstruction.
- We use AutoProtoNet to visualize prototypes and find that they are comparable in quality to those of an autoencoder. AutoProtoNet also remains accurate on few-shot image classification benchmarks.

Submitted to the 5th Workshop on Meta-Learning at NeurIPS 2021, Sydney, Australia. Do not distribute.

- We devise a prototype refinement method, which can be used to debug inadequate prototypes, and we validate the performance of the resulting model using a novel validation set of in-the-wild images.

Our goal in this work is to elucidate the benefits of learning embeddings that can be visualized and interpreted by humans. To the best of our knowledge, there is no meta-learning approach that allows for a human to play a role in the fine-tuning of the base model.

2 Related Work

2.1 Meta-learning and Prototypical Networks

Before meta-learning, transfer learning was used to handle few-shot problems. In transfer learning, a feature extractor is trained on a large dataset, then fine-tuned for new tasks [2]. However, transfer learning has some drawbacks. For example, adding a new class may require re-training the model and, in the few-shot setting, overfitting few example images is possible.

Meta-learning algorithms aim to learn a "base" model that can be quickly fine-tuned for a new task. The base model is trained using a set of training tasks $\{\mathcal{T}_i\}$, sampled from some task distribution. Each task consists of *support* data, $\mathcal{T}_i^s$, and *query* data, $\mathcal{T}_i^q$. Support data is used to fine-tune the model, while query data is used to evaluate the resulting model. Practically speaking, each task is an image classification problem involving only a small number of classes. The number of examples per class in the support set is called the *shot*, and the number of classes is called the *way*. For example, in 5-way 1-shot learning, we are given 1 example for each of the 5 classes to use for fine-tuning.

Following the meta-learning framework presented in [8], Algorithm 1 can be used as a general way to understand both metric-learning methods [23] [20] and gradient-based methods like MAML [6].

Algorithm 1 The meta-learning framework

Input: Base model, F_θ
Input: Fine-tuning algorithm, A
Input: Learning rate, γ
Input: Distribution over tasks, $p(\mathcal{T})$

Initialize θ, the weights of F
while not done **do**
    Sample batch of tasks $\{\mathcal{T}_i\}_{i=1}^n$, where $\mathcal{T}_i \sim p(\mathcal{T})$ and $\mathcal{T}_i = (\mathcal{T}_i^s, \mathcal{T}_i^q)$
    for i=1,...,n **do**
        $\theta_i \leftarrow A(\theta, \mathcal{T}_i^s)$ ▷ Fine-tune model on $\mathcal{T}_i^s$ (inner loop)
        $g_i \leftarrow \nabla_\theta \mathcal{L}(F_{\theta_i}, \mathcal{T}_i^q)$
    end for
    $\theta \leftarrow \theta - \frac{\gamma}{n}\sum_i g_i$ ▷ Update base model parameters (outer loop)
end while

For ProtoNets [20], the base model $F_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^M$ is an embedding network which takes an image $x \in \mathbb{R}^D$ as input and outputs an embedding vector of dimension M. Suppose, for example, we have a K-way task $\mathcal{T}_i = (\mathcal{T}_i^s, \mathcal{T}_i^q)$ where $\mathcal{T}_i^s = \{(x_{i,1}, y_{i,1}), (x_{i,2}, y_{i,2}), ..., (x_{i,N}, y_{i,N})\}$, and where $y_{i,j} \in \{1, ..., K\}$. Additionally, let $S_k \subset \mathcal{T}_i^s$ denote the set of support examples of class k. Then, a prototypical network computes a prototype p_k for each class k by computing a class-wise mean of embedded support examples:

$$p_k = \frac{1}{|S_k|} \sum_{(x,y)\in S_k} F_\theta(x) \tag{1}$$

Thus, in the case of ProtoNets, the fine-tuning algorithm A does not update model parameters θ, but instead it computes a set of prototypes which the base model will use to classify query data. We can think of A as a function taking both embedding network parameters θ and support data $\mathcal{T}_i^s$ and returning a tuple θ_i consisting of a set of prototypes and an unchanged set of model parameters; i.e., $A(\theta, \mathcal{T}_i^s) = (\{p_k\}_{i=0}^k, \theta) = \theta_i$. In this way, F_{θ_i} in Algorithm 1 refers to using the base model parameters θ and the set of prototypes $\{p_k\}_{i=0}^{k}$ during inference. Given a distance function $d : \mathbb{R}^M \times \mathbb{R}^M \to [0, \infty)$ and a query point x, a ProtoNet produces a distribution over classes based on a softmax over distances to the prototypes in embedding space:

$$p_\theta(y = k|x) = \frac{\exp(-d(F_\theta(x), p_k))}{\sum_{k'} \exp(-d(F_\theta(x), p_{k'}))} \tag{2}$$

Training proceeds by minimizing the negative log-likelihood $\mathcal{L}(\theta) = -\log p_\theta(y = k|x)$ of the true class k using SGD. Unfortunately, ProtoNet does not provide a way to understand the embedding space or visualize p_k – a problem we directly address in this work.

2.2 Understanding Meta-learning Approaches

Investigating the ability of meta-learning methods to adapt to new tasks has been the subject of numerous studies. The success of meta-learning approaches certainly seems to suggest that the representations learned by meta-learning must be different than those learned through standard training [9]. Goldblum et al. [9] find that meta-learned feature extractors outperform classically trained models of the same architecture and suggest that meta-learned features are qualitatively different from conventional features. While work has been done to understand how the meta-learning networks train [10][7], there has been little to no focus on developing tools to interpret the meta-learned models.

2.3 Interpretability in Convolutional Models

In safety or security-critical applications, understanding why a classification system made a certain prediction is important. Just because a classification system is highly accurate, does not mean the network has learned the right kinds of features [11]. We believe that a system that can demonstrate its logic semantically or visually is more likely to be trusted and used. Being that a ProtoNet is primarily a convolutional neural network, it is appropriate to understand progress on interpretability of convolutional neural networks (CNN).

There are many research branches within the umbrella of CNN interpretability including visualizations of intermediate network layers [25][16][19][21], diagnosis of CNN representations [27][26], and building explainable models [28]. In contrast to works which focus their attention on CNN layers and activations, we take a more specific approach in visualizing embedding space for ProtoNets.

Zhang et al. [28] propose a compelling method of modifying convolutional layers so that each filter learns to represent a particular object part, thus allowing for each filter to correspond to a semantically meaningful image feature. We believe there could be interesting work incorporating this technique into meta-learning approaches, but is not appropriate for a shallow embedding network like the one we employ for ProtoNets.

2.4 Generative Models

Work on Variational Prototyping Encoder (VPE) [12] is most similar to ours in that a meta-task is used to learn an embedding space suitable for both few-shot learning and unseen data representation. In contrast, we do not focus on the image translation task from real images to prototypes and instead focus our attention on visualizing prototypes for interpretability and refinement.

There are also a number of works which investigate connections between autoencoder architectures and meta-learning, but which are not directly applicable for interpretability of few-shot image classification. For example, Wu et al. [24] propose the Meta-Learning Autoencoder (MeLA) framework which learns a recognition and generative model to transform a single-task model into one that can quickly adapt to new tasks using few examples. However, their framework is meant for the more general understanding of *tasks* like physical state estimation and video prediction, as opposed to the image classification tasks which we focus on. Similarly, Epstein et al. [5] develop a meta-learning framework consisting of joint autoencoders for the purpose of learning multiple tasks simultaneously, but this approach is tailored more for the field of multi-task learning.

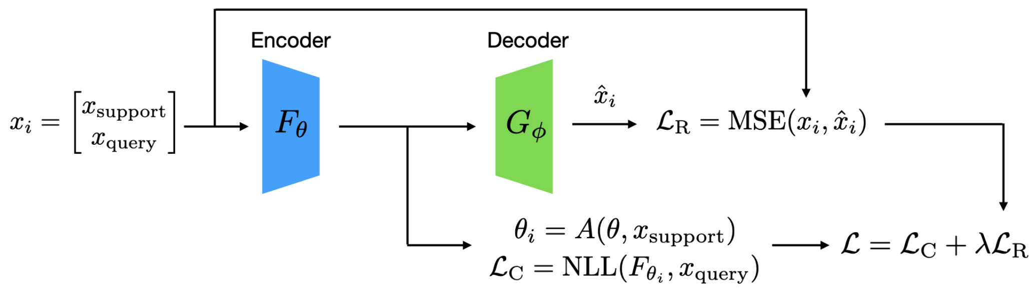


Figure 1: Visualization of the forward pass through AutoProtoNet.

3 Algorithm

Our interpretability algorithm takes advantage of the simplicity of the ProtoNet classification method. In particular, a ProtoNet classifies query data according to the class of the prototype which the query data's embedding is nearest to, typically in Euclidean space. This classification method raises an obvious question: what does a prototype look like? To answer this question, we extend ProtoNets with a decoder to reconstruct images from embeddings.

3.1 Data

The CIFAR-FS dataset [3] is a recent few-shot image classification benchmark consisting of all 100 classes from CIFAR-100 [14]. Classes are randomly split into 64, 16, and 20 for meta-training, meta-validation, and meta-testing respectively. Every class contains 600 images of size 32×32.

The *mini*ImageNet dataset [23] is another standard benchmark for few-shot image classification. It consists of 100 randomly chosen classes from ILSVRC 2012 [4], which are split into 64, 16, and 20 classes for meta-training, meta-validation, and meta-testing respectively. For every class, there are 600 images of size 84×84. We adopt the commonly-used Ravi and Larochelle split proposed in [18].

3.2 Architecture

AutoProtoNet consists of an encoder-decoder architecture which compresses the input to produce an embedding which must be reconstructed by the decoder. There 4 sequential convolution blocks for the encoder and 4 sequential transpose convolution blocks for the decoder. The details of these blocks can be found in Table 2 of Appendix B. A forward pass through the model is shown in Figure 1.

Output padding is used in the second transpose convolution block of the decoder to ensure that the output size of the final transpose convolution block matches the input 84×84 dimensions of *mini*ImageNet images, but no output padding modifications are necessary for CIFAR-FS images.

Our architectural design choices imply that a 84×84 *mini*ImageNet image is embedded as 1600-dimensional vector, while a 32×32 CIFAR-FS image is embedded as 256-dimensional vector.

3.3 Training

Training AutoProtoNet is not much different from training a ProtoNet. The main difference is that we augment the meta-training loop with a reconstruction loss to regularize the embedding space and make it suitable for image reconstruction. We display the forward pass through AutoProtoNet in Figure 1 and adapt the meta-learning framwork from Section 2.1 to describe the meta-training of AutoProtoNet in Algorithm 2.

Our "base" model now consists of parameters ψ which is a concatenation of encoder network parameters θ and decoder network parameters ϕ. In Line 5 of Algorithm 2, we pass both support and query data from the current task $\mathcal{T}_i$ through the encoder and decoder to produce a reconstruction $\hat{\mathcal{T}}_i$. This reconstruction is then compared to the original data using mean squared error (MSE) loss.

Algorithm 2 AutoProtoNet Meta-Learning

Input: Encoder and decoder networks, F_θ and G_ϕ, where $\psi = [\theta; \phi]$
Input: Fine-tuning algorithm, A
Input: Reconstruction loss weight, λ
Input: Learning rate, γ
Input: Distribution over tasks, $p(\mathcal{T})$

1: Initialize θ, ϕ, the weights of encoder and decoder
2: **while** not done **do**
3:     Sample batch of tasks $\{\mathcal{T}_i\}_{i=1}^n$, where $\mathcal{T}_i \sim p(\mathcal{T})$ and $\mathcal{T}_i = (\mathcal{T}_i^s, \mathcal{T}_i^q)$
4:     **for** i=1,...,n **do**
5:         $\hat{\mathcal{T}}_i \leftarrow G_\phi(F_\theta(\mathcal{T}_i))$ ▷ Reconstruct task data
6:         $\mathcal{L}_R \leftarrow \mathrm{MSE}(\mathcal{T}_i, \hat{\mathcal{T}}_i)$ ▷ Compute reconstruction loss
7:         $\theta_i \leftarrow A(\theta, \mathcal{T}_i^s)$ ▷ Compute prototypes (inner loop)
8:         $\mathcal{L}_C \leftarrow \mathrm{NLL}(F_{\theta_i}, \mathcal{T}_i^q)$ ▷ Compute classification loss
9:         $\mathcal{L} \leftarrow \mathcal{L}_C + \lambda \mathcal{L}_R$
10:         $g_i \leftarrow \nabla_\psi \mathcal{L}$
11:     **end for**
12:     $\psi \leftarrow \psi - \frac{\gamma}{n} \sum_i g_i$ ▷ Update base model parameters (outer loop)
13: **end while**

The finetuning algorithm in Line 7 of Algorithm 2 is identical to the description in Section 2.1, where $\theta_i = (\{p_k\}_{i=0}^k, \theta)$ is a tuple consisting of a set of prototypes for every class and the encoder network's model parameters. Both of these are used to compute the likelihood of the true labels of our query data as in Equation 2, which is maximized by minimizing the negative log-likelihood (NLL). Finally, the classification loss $\mathcal{L}_C$ and the reconstruction loss $\mathcal{L}_R$ are summed so they can be jointly optimized.

We meta-train ProtoNet and AutoProtoNet on both *mini*ImageNet and CIFAR-FS. To create a prototype reconstruction baseline, we also train two models which make use of ILSVRC 2012 [4], which we refer to as ImageNet Autoencoder and ImageNet AutoProtoNet. Note that because *mini*ImageNet is a subset of ILSVRC 2012, the ImageNet models also provide insight into whether more data during pretraining offers any benefit for meta-learning or prototype reconstructions. All training was performed on a single NVIDIA Quadro P6000 from our internal cluster. Training details for each model used in this work are described below.

ProtoNet Using Algorithm 1, we meta-train a standard ProtoNet for 30 epochs using SGD. Our SGD optimizer uses Nesterov momentum of 0.9, weight decay of 5×10^{-4}, and a learning rate of 0.1, which we decrease to 0.06 after 20 epochs.

AutoProtoNet Using Algorithm 2, we meta-train an AutoProtoNet for 30 epochs using SGD. We use the same SGD settings as in ProtoNet training. We use a reconstruction loss weight $\lambda = 1$. Following [20], both ProtoNet and AutoProtoNet models were trained using 20-way 5-shot episodes, where each class contains 15 query points per episode, for 30 epochs.

ImageNet Autoencoder We train an autoencoder of the same architecture as AutoProtoNet using only mean squared error (MSE) loss on ILSVRC 2012 [4] for 20 epochs. We use the SGD optimizer with Nesterov momentum of 0.9, weight decay of 5×10^{-4}, and a learning rate of 0.1, which we decrease by a factor of 10 every 5 epochs. To evaluate this model's performance on benchmark few-shot image classification datasets, we make use of the only the encoder to produce embeddings and produce classification labels using the standard ProtoNet classification rule.

ImageNet AutoProtoNet We use the encoder and decoder weights from the ImageNet Autoencoder as a starting point for the weights of an AutoProtoNet. All other training details are identical to that of AutoProtoNet, which we meta-train using Algorithm 2.

The 5-way 5-shot test set accuracies of all models used in this work are shown in Table 1. AutoProtoNet is able to maintain the same level of few-shot image classification accuracy on benchmark datasets as a standard ProtoNet. While we expected AutoProtoNet to have an advantage due to

Table 1: 5-way 5-shot test set accuracies with 95% confidence intervals.

Model	*mini*ImageNet	CIFAR-FS
ImageNet Autoencoder	$36.83 \pm 0.48\%$	$46.08 \pm 0.58\%$
ImageNet AutoProtoNet	$70.76 \pm 0.51\%$	$79.65 \pm 0.52\%$
ProtoNet	$70.20 \pm 0.52\%$	$80.31 \pm 0.51\%$
AutoProtoNet	$70.61 \pm 0.52\%$	$80.16 \pm 0.52\%$

having to incorporate features useful for reconstruction into embeddings, our results suggest that these reconstruction features are not always useful. Given the additional ILSVRC 2012 [4] data during pretraining, we also expected that ImageNet AutoProtoNet would outperform all other models, but our test results demonstrate that representations learned for image reconstruction are not too helpful for few-shot image classification. Test set accuracies for ImageNet Autoencoder underscore the point that an embedding space trained for only reconstruction is by no means competitive for few-shot classification, though it does achieve better than chance accuracy.

4 Experiments

4.1 Prototype Visualization

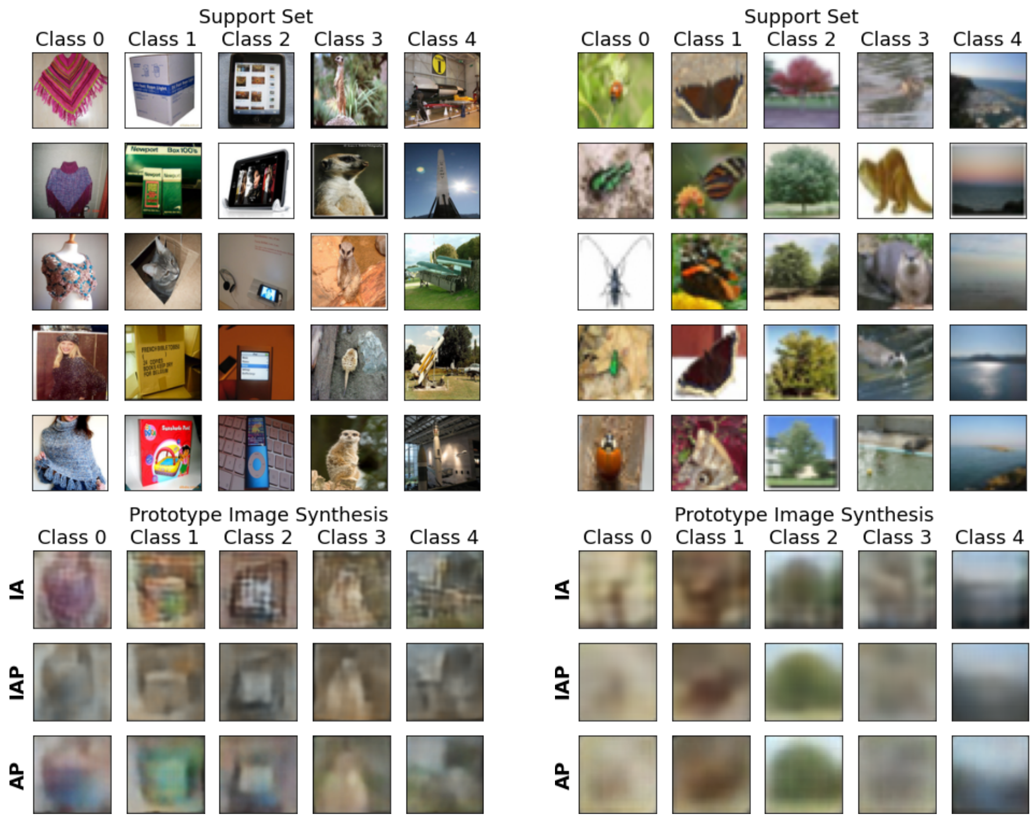
While a standard ProtoNet employs an intuitive nearest-neighbor classification rule for query points, there is no intuitive way for a user to understand what a prototype embedding represents. Prototypical embeddings are crucial to understanding the decision boundaries of ProtoNets. The idea is that a ProtoNet embeds similar images nearby in embedding space, but without a way to visualize these embeddings, we argue that a human practitioner would be unable to debug or improve their deployed model. AutoProtoNet addresses this issue by learning an embedding space that is suitable for image reconstruction.

Figure 2 displays prototype visualizations given a validation support set from *mini*ImageNet and CIFAR-FS. The ImageNet Autoencoder (**IA**) and ImageNet AutoProtoNet (**IAP**) were both pretrained on all of ILSVRC 2012 [4], and so classes present in this validation support set are not novel classes because *mini*ImageNet is a subset of ILSVRC 2012. However, in the case of the AutoProtoNet (**AP**), the classes in this validation support set are novel and the synthesized prototype images remain qualitatively on-par with the models trained with more data (such as ImageNet Autoencoder), suggesting that meta-tasks during training were sufficient to regularize an embedding space suitable for image synthesis. Analyzing the prototype reconstructions from CIFAR-FS in Figure 2(b), we see that prototype visualizations are generally too blurry to help a human determine whether the model has learned a sufficient representation of a class. We believe part of the problem is the low resolution and size of CIFAR-FS images.

4.2 Human-guided Prototype Refinement

To highlight the benefits of an embedding space suitable for image reconstruction, we designed an experiement to demonstrate how a human can guide prototype selection at test-time using AutoProtoNet. Assuming the user knows the kinds of images the model will encounter at inference time and given the ability to capture one more image, could we refine an initial prototype to achieve higher accuracy on the validation set?

Data Collection Based on objects we had around the house, we chose to formulate a 5-way 1-shot classification problem between "door knob", "frying pan", "light switch", "orange", and "water bottle". Note that "orange" and "frying pan" are classes in the *mini*ImageNet training split, but all other classes are novel. Because we sought to demonstrate how one might use an AutoProtoNet in a real-world setting, all 55 images in this task are novel, in-the-wild images, captured using an iPhone 12. Our support set consists of 5 images (1 image per class). Our validation set consists of 50 images (10 images per class) and can be found in Figure 4 of Appendix A.



(a) *mini*ImageNet (b) CIFAR-FS

Figure 2: Support sets for a 5-way 5-shot validation task of *mini*ImageNet (a) and CIFAR-FS (b). The embeddings of every image within a class are averaged to form a prototype embedding which is then synthesized as an image by using the decoder of an ImageNet Autoencoder (**IA**), an ImageNet AutoProtoNet (**IAP**), and an AutoProtoNet (**AP**).

Prototype Refinement Prototype refinement is a debugging technique meant for cases in which a human believes prototype visualization may not be representative of the class. To exaggerate the idea of prototype refinement, we purposefully choose the back-side of a frying pan as a support image for class 1 ("frying pan") so that the prototype visualization has undesirable image features. Generally, a prototype for an arbitrary object of a novel class is likely to be visually ambiguous if the embedding network did not train on a suitable dataset, so this setup is conceivable in the real-world.

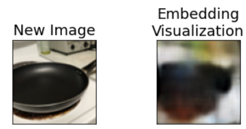
For our classification model, we make use of the AutoProtoNet described in Section 3.3. To apply AutoProtoNet to this new classification task, we "fine-tine" AutoProtoNet by providing a support set shown in Figure 3(a). After meta-learning, an AutoProtoNet's only changeable parameters are its prototypes which, by design, can be reconstructed into images using the decoder. By visually understanding an AutoProtoNet's embedding space, a user can choose to change image features of a prototype reconstruction, thus changing the prototype itself. In contrast, a standard ProtoNet performs inference using its support data, which is visually inaccessible and uninterpretable.

Using a newly captured image $x \in \mathbb{R}^d$, we use the encoder F_θ to generate an embedding $p = F_\theta(x)$. Given an initial prototype p_k for class k, we use the decoder G_ϕ to synthesize images $\hat{x}_i \in \mathbb{R}^d$ for interpolations between p_k and p as follows:

$$\hat{x}_i = G_\phi((1-\alpha)p_k + \alpha p) \qquad \alpha \in [0,1] \tag{3}$$

Results Using the initial prototypes from Figure 3(a), AutoProtoNet achieves 80% accuracy on the validation set consisting of 50 images from all 5 classes. The 10 misclassified images are all of the

(a) Support set and prototype visualizations



(b) New image and corresponding embedding

(c) Interpolating 10 steps from initial prototype to new image embedding

(d) New set of prototypes

Figure 3: Steps for human-guided prototype selection in a 5-way 1-shot task. Step (a): a human chooses an initial prototype to refine. Step (b): a human captures one additional image to guide prototype refinement. Step (c): Interpolations between the initial prototype and the new image embedding (index 9) are shown to the human and a new prototype selection is made. Step (d): A new set of prototypes is set, with class 2 having been refined.

"frying pan" class. After debugging the "frying pan" prototype by capturing an additional image of a correctly-oriented frying pan and choosing an interpolation, the resulting embedding is used as the new support as shown in Figure 3(d). Under the new human-guided prototypes, AutoProtoNet achieves an accuracy of 98% on the validation set, where the single misclassified image is of the "door knob" class.

The novelty of our method lies in the ability for a human to fine-tune the model in an interactive way, leading to a performance increase in validation set accuracy. In this example, AutoProtoNet's decoder allowed for the visualization of the prototype embedding, which we found to be visually incorrect. Thus, we captured an additional, more representative image to designate the direction in which to move the initial prototype to fit a human-designated criteria.

5 Conclusion

With AutoProtoNet, we present a step toward meta-learning approaches capable of giving some insight into their learned parameters. We argue that if meta-learning approaches are to be useful in practice, there should be ways for a human to glean some insight into why a classification might have been made. Through prototype visualizations and a prototype refinement method, we highlight the benefits of AutoProtoNet and take steps to improve a simple few-shot classification algorithm by making it more interpretable while maintaining the same degree of accuracy as a standard ProtoNet.

Our proposed method could likely be extended to Relation Networks [22], MetaOptNet [15], or R2D2 [3], with a decoder network to visualize embeddings. It may also be possible to meta-train a variational autoencoder to learn a latent space more suitable for detailed image synthesis. We believe generative models can play a larger role in interpretability of meta-learning algorithms.

To confirm the effectiveness of our interpretability results, we intend to perform a human subjects study where a human determines whether prototype visualizations help in understanding classification results. We also recognize the limits of using a small dataset to evaluate the performance of our prototype refinement method. We leave the creation of a larger, more diverse validation set to future work.

References

[1] H. Altae-Tran, B. Ramsundar, A. S. Pappu, and V. S. Pande. Low data drug discovery with one-shot learning. *CoRR*, abs/1611.03199, 2016. URL http://arxiv.org/abs/1611.03199.

[2] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. JMLR Workshop and Conference Proceedings, 2012.

[3] L. Bertinetto, J. F. Henriques, P. H. S. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. *CoRR*, abs/1805.08136, 2018. URL http://arxiv.org/abs/1805.08136.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[5] B. Epstein, R. Meir, and T. Michaeli. Joint autoencoders: A flexible meta-learning framework. In M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 494–509, Cham, 2019. Springer International Publishing. ISBN 978-3-030-10925-7.

[6] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

[7] N. Frosst, N. Papernot, and G. Hinton. Analyzing and improving representations with the soft nearest neighbor loss, 2019.

[8] M. Goldblum, L. Fowl, and T. Goldstein. Robust few-shot learning with adversarially queried meta-learners. *CoRR*, abs/1910.00982, 2019. URL http://arxiv.org/abs/1910.00982.

[9] M. Goldblum, S. Reich, L. Fowl, R. Ni, V. Cherepanova, and T. Goldstein. Unraveling meta-learning: Understanding feature representations for few-shot tasks. *CoRR*, abs/2002.06753, 2020. URL https://arxiv.org/abs/2002.06753.

[10] W. R. Huang, Z. Emam, M. Goldblum, L. Fowl, J. K. Terry, F. Huang, and T. Goldstein. Understanding generalization through visualizations. *CoRR*, abs/1906.03291, 2019. URL http://arxiv.org/abs/1906.03291.

[11] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. Adversarial examples are not bugs, they are features, 2019.

[12] J. Kim, T. Oh, S. Lee, F. Pan, and I. S. Kweon. Variational prototyping-encoder: One-shot learning with prototypical images. *CoRR*, abs/1904.08482, 2019. URL http://arxiv.org/abs/1904.08482.

[13] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby. Large scale learning of general visual representations for transfer. *CoRR*, abs/1912.11370, 2019. URL http://arxiv.org/abs/1912.11370.

[14] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[15] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. *CoRR*, abs/1904.03758, 2019. URL http://arxiv.org/abs/1904.03758.

[16] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. *CoRR*, abs/1412.0035, 2014. URL http://arxiv.org/abs/1412.0035.

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[18] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

[19] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

[20] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. *CoRR*, abs/1703.05175, 2017. URL `http://arxiv.org/abs/1703.05175`.

[21] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net, 2015.

[22] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. *CoRR*, abs/1711.06025, 2017. URL `http://arxiv.org/abs/1711.06025`.

[23] O. Vinyals, C. Blundell, T. P. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016. URL `http://arxiv.org/abs/1606.04080`.

[24] T. Wu, J. Peurifoy, I. L. Chuang, and M. Tegmark. Meta-learning autoencoders for few-shot prediction, 2018.

[25] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL `http://arxiv.org/abs/1311.2901`.

[26] Q. Zhang, R. Cao, F. Shi, Y. N. Wu, and S.-C. Zhu. Interpreting cnn knowledge via an explanatory graph, 2017.

[27] Q. Zhang, R. Cao, Y. N. Wu, and S.-C. Zhu. Growing interpretable part graphs on convnets via multi-shot learning, 2017.

[28] Q. Zhang, Y. N. Wu, and S.-C. Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In Section 4.1 and in the conclusion Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We believe there are no negative impacts since we use already publicly existing work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In Appendix C.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Training details outlined in Section 3.3.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Described in Section 3.3.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] In Appendix C.
(b) Did you mention the license of the assets? [Yes] In Appendix C.
(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] In Appendix C.
(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] In Appendix C.
(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Data does not contain identifiable information.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Validation Set for Custom Classification Task

In Figure 4, we display the 50 images of our custom 5-way validation set. The images from the "light switch" and "door knob" classes are diverse in terms of shape, pose, and lighting condition.

Figure 4: Validation set for experiment described in Section 4.2

B Architecture Details

In our description of the AutoProtoNet architecture in Table 2, we display output sizes for the first Conv Block of the encoder and the first Conv Transpose Block of the decoder, assuming an 84×84 *mini*ImageNet image is used as input.

Table 2: AutoProtoNet Architecture Components

Conv Block			Conv Transpose Block		
Layer	Parameters	Output Size	Layer	Parameters	Output Size
Conv	3×3, 64	$64 \times 84 \times 84$	Conv Transpose	2×2, $*2$	$64 \times 10 \times 10$
Batch Norm			Batch Norm		
Max Pool	3×3, /2	$64 \times 42 \times 42$	Conv	3×3, 64	$64 \times 10 \times 10$

C Implementation Details

We use PyTorch [17] and work on a fork of code used for [8], which uses the MIT License. Our fork can be used to reproduce experiments and is available here: REDACTED.