UNDER REVIEW

RESCIENCE C

Editorial

[Re]Simplified Training for Prototype Based Models

Clifford Wilmot[1, ID] and Michael Panchenko[1, ID]

[1]appliedAI Initiative GmbH, Freddie-Mercury-Straße 5, 80797 München

Edited by
(Editor)

Received
–

Published
–

DOI
–

Abstract — *TODO: Check how this should be written. This likely depends on whether we're presenting the paper as pure, isolated reproductions of two different papers, or as an extension and simplification that's applicable to several papers in the area (the latter is more accurate in my view).*
Image classification models based upon similarity comparisons with learnable prototype tensors have arisen as a promising approach to achieving interpretability. However, quite a few works in this area have employed complex, custom optimization algorithms and manual hyperparameter tuning in order achieve accurate results. In this paper, we reimplement a couple of major papers in this area in a single, modular codebase, and confirm their key results. We then show that their models can be successfully trained with more standard algorithms and hyperparameters, sometimes even achieving superior accuracy. We also briefly investigate negative prototypes and perform a qualitative analysis of the interpretability of the models.

1 Introduction

In the last several years, machine learning models have seen widespread success in computer vision, with applications in many areas such as healthcare, agriculture, and manufacturing. In one subfield of computer vision — image classification —, deep learning models have surpassed human-level performance in many specific tasks. However, these models have often been critised for their "black box" nature, making it hard to trust their predictions, or to understand the circumstances under which they might fail.
In order to increase the interpretability of these models, many different approaches have been proposed. Some are based on taking a trained, uninterpretable model, and performing some form of analysis to gain insights into how it works; other works have instead tried to build models that are interpretable from the start. One relatively new case of the latter method is prototype based models; with this approach, parts of images are compared to a collection of tensors — known as prototypes — which are learned during training, classification then proceeds based on the outcomes of these comparisons. Many prototype-based models have been able to achieve accuracy that's competitive with black box models, but there have also been several criticisms levelled at them. Firstly, they have often used complex, custom optimization algorithms and manual hyperparameter tuning. Secondly, both works exclusively use "positive" prototypes, in which we seek parts of images that are most similar to the prototypes; papers such as [1] have discussed trying to avoid "negative" reasoning — in which we look for parts of images most different to any prototypes —, but there has been little quantitative analysis and justification for this. Finally, several works have questioned how interpretable the models truly are.

Correspondence should be addressed to Clifford Wilmot (c.wilmot@appliedai.de)
The authors have declared that no competing interests exist.
Code is available at https://github.com/rescience-c/template..

In this work, we build upon two major papers in the area of prototype-based image classification: "This looks like that: deep learning for interpretable image recognition" by Chen et al,[2] and "Neural prototype trees for interpretable fine-grained image recognition" by Nauta et al.[3] Our contribution comprises several parts:

- We reimplement both works in a single, modular codebase in Python 3 with PyTorch; our reimplementation was originally forked from the authors' open sourced code for [3] at https://github.com/M-Nauta/ProtoTree, but we have performed sufficient refactoring that our code is almost entirely new.
- We successfully replicate the core accuracy claims from these papers. We show that by structuring the training and modelling in a suitable manner, it's feasible to successfully train the models with standard deep learning optimization algorithms and hyperpameters.
- We demonstrate that negative prototype models suffer far worse performance than positive ones, and discuss the implications of this.
- We perform a qualitative assessment of the interpretability of the models by looking at the outputs of visualization code we have written.

2 Methodology

In this section, we discuss the models themselves, and the optimization algorithms that were used to train them in the original papers. We also discuss our use of alternative, simpler, more standard optimization procedures. While our changes may of course reduce accuracy, we believe that this approach will aid in understanding how well prototype-based models work "out-of-the-box", which is more representative of what users of these models can expect to achieve in practice.

2.1 Models

Both models consist of a common "base" section that contains the prototypes, followed by a model-specific section that takes the outputs from the base section and produces the classifications.

Base — The base section starts with a typical feature-extraction backbone convolutional neural network with output shape $H \times W \times D_c$; it is usually pretrained on an image dataset such as ImageNet. This is followed by an "addon" section for more prototype-specific feature extraction; the two papers and open sourced codebases contain several variants on this, but for our paper we opt for the following 4 operations in order:

1. 1×1 convolutional layer with D_p output channels, where D_p is the number of prototypes
2. ReLU
3. 1×1 convolutional layer with D_p output channels
4. Sigmoid (we suspect that ensuring that outputs lie in the $[0,\ 1]$ interval helps to give sensible results in the prototype similarity calculations)

After the addons, we have a prototype layer consisting of D_p different $H_p \times W_p$ prototype tensors $\mathbf{p}_1, \cdots, \mathbf{p}_j, \cdots, \mathbf{p}_J$, with $H_p < H$ and $W_p < W$. We then perform a form of

generalized convolution, in which each $\mathbf{p}_j$ is treated as a kernel that we slide over each patch $\tilde{z}$ in an image z. In the positive prototype case, we compute

$$s_j = \min_{\tilde{z} f(\tilde{z},\, \mathbf{p}_j)},$$

for each j, where f is a chosen similarity function; in the negative prototype case, we instead compute $\max_{\tilde{z}}$. In this paper and the original papers $f(\tilde{z},\, \mathbf{p}_j) = |\tilde{z} - \mathbf{p}_j|_2$. In practice, efficiently computing f can result in square rooting negative numbers in $\sqrt{|\tilde{z} - \mathbf{p}_j|_2^2}$ due to numerical instabilities. The two original papers dealt with this by instead computing $\sqrt{|\tilde{z} - \mathbf{p}_j|_2^2 + \varepsilon}$, but we found that that

2.2 Optimization

The original optimization algorithm for ProtoPNet consists of 3 main steps:

1.

2.3 Hyperparameters

No attempt was made to manually tune the hyperparameters for each dataset. The Adam optimizer uses a learning rate of 10^{-5} for the backbone network (excluding the final layer), and 10^{-3} for all other parameters; no learning rate decay is performed; $\beta_1 = 0.999$, $\beta_2 = 0.9$, and eps is 10^{-7}. All prototypes are spatial size 1×1 with a depth of 256. We use a batch size of 64. We train for 100 epochs, and the backbone network (excluding the final layer) is frozen for the first 30 epochs.
The ProtoTree depth is 9 (so 512 prototypes), and for ProtoPNet we have 10 prototypes per class in the dataset.
The backbone network is ...

2.4 Code

Both [2] and [3] have open sourced their code in https://github.com/cfchen-duke/ProtoPNet and https://github.com/M-Nauta/ProtoTree, respectively. These implementations are both written in Python 3 using PyTorch. We found that both codebases tended to couple together the model, optimization, and training loop code; this was fine for those papers, but made it challenging for us to make substantial changes to any of these components. We therefore forked https://github.com/M-Nauta/ProtoTree and refactored it into a form that allowed us to combine it with [2] and conduct our investigations; at this point there have been so many changes to the code that our implementation is more akin to a complete rewrite than a refactor.
Our code is still in Python 3, but it now uses PyTorch Lightning instead of just PyTorch. We have a `ProtoBase` class which handles the prototypes and similarity calculations. The two models are implemented in the `ProtoPNet` and `ProtoTree` class; they use `ProtoBase` via composition, and the optimization algorithms for the models are included in methods on the classes. We hope that this approach will allow other models and forms of prototypes to be implemented by slotting in new model classes and prototype classes, respectively, instead of needing widespread changes to disparate parts of the codebase.

2.5 Data

We ran our experiments on the CUB-200-2011 dataset with 200 bird species (CUB), and the Stanford Cars dataset with 196 car types (CARS).[4][5] These were the same datasets used by both [2] and [3].
The data were augmented with ...
As in [3], the CUB-200-2011 dataset was further augmented with corner...

3 Results

3.1 Hardware Used

All experiments were run on a virtual machine with a 32GB VRAM NVidia GPU, 8 Intel ..., and 64GB RAM. (TODO: details)

3.2 Data

aaa

References

1. J. Donnelly, A. J. Barnett, and C. Chen. "Deformable protopnet: An interpretable image classifier using deformable prototypes." In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. 2022, pp. 10265–10275.
2. C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. K. Su. "This looks like that: deep learning for interpretable image recognition." In: **Advances in neural information processing systems** 32 (2019).
3. M. Nauta, R. Van Bree, and C. Seifert. "Neural prototype trees for interpretable fine-grained image recognition." In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. 2021, pp. 14933–14943.
4. J. Krause, M. Stark, J. Deng, and L. Fei-Fei. "3d object representations for fine-grained categorization." In: **Proceedings of the IEEE international conference on computer vision workshops**. 2013, pp. 554–561.
5. C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. "The caltech-ucsd birds-200-2011 dataset." In: (2011).