CaBRNet, An Open-Source Library For Developing And Evaluating Case-Based Reasoning Models

Romain **Xu-Darme**[1,*], Aymeric **Varasse**[1], Alban **Grastien**[1], Julien **Girard-Satabin**[1] and Zakaria **Chihani**[1]

[1]Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Abstract

In the field of explainable AI, a vibrant effort is dedicated to the design of self-explainable models, as a more principled alternative to *post-hoc* methods that attempt to explain the decisions after a model opaquely makes them. However, this productive line of research suffers from common downsides: lack of reproducibility, unfeasible comparison, diverging standards. In this paper, we propose CaBR-Net, an open-source, modular, backward-compatible framework for **Ca**se-**B**ased **R**easoning **Net**works: https://github.com/aiser-team/cabrnet.

Keywords

Explainable AI, Case-based reasoning, Computer vision, Evaluation

1. Introduction

As a reflection of the social and ethical concerns related to the increasing use of AI-based systems in modern society, the field of explainable AI (XAI) has gained tremendous momentum in recent years. XAI mainly consists of two complementary avenues of research that aim at shedding some light into the inner-workings of complex ML models. On the one hand, *post-hoc* explanation methods apply to existing models that have often been trained with the sole purpose of accomplishing a given task as efficiently as possible (*e.g.,* accuracy in a classification task). On the other hand, self-explainable models are designed and trained to produce their own explanations along with their decision. The appeal of self-explainable models resides in the fact that rather than using an approximation (*i.e.,* a *post-hoc* explanation method) to understand a complex model, it is better to directly enforce a simpler (and more understandable) decision-making process during the design and training of the ML model, provided that such a model would exhibit an acceptable level of performance.

In the case of image classification, works such as ProtoPNet [1] and subsequent improvements (ProtoTree [2], ProtoPool [3], ProtoPShare [4], TesNet [5], PIP-Net [6]) have shown that self-explainable models can display accuracy results on par with more opaque state-of-the-art

architectures. Such models are based on the principle of *case-based reasoning* (CBR), in which new instances of a problem (the classification task) are solved using comparisons with an existing body of knowledge that takes the form of a database of prototypical representations of class instances. In other words, the classification of an object is explained by the fact that it *looks like* [1] another object from the training set for which the class is known. CBR classifiers undeniably represent a stepping stone towards more understandable models. Yet, several recent works [7, 8, 9, 10, 11] have questioned the interpretability of such models, proposing several evaluation metrics to go beyond the mere accuracy of the model and to estimate the quality of the explanations generated. In practice however, a systematic comparison between different proposals for CBR models using these different metrics can be difficult to carry out in the absence of a unified framework that could gather the resources provided by their respective authors (*e.g.,* source codes made available on GitHub).

In this context, we propose *CaBRNet* (**Ca**se-**B**ased **R**easoning **Net**works), an open-source Py-Torch library that proposes a generic approach to CBR models. CaBRNet focuses on modularity, backward compatibility, and reproducibility, allowing us to easily implement existing works from the state of the art, propose new and innovative architectures, and compare them within a unified framework using multiple dedicated evaluation metrics.

2. CaBRNet: One framework to rule them all...

In a field of research as wide as XAI, where the proliferation of tools and methods is increasing, seeking a common framework is a natural temptation that often leads to yet another tool that evolves besides the rest. To minimize this risk, CaBRNet is designed with three main objectives, essential for a lasting acceptance: supporting past state-of-the-art methods through backward compatibility, facilitating present developments by striving for modularity, and ensuring their reusability in future works through reproducibility.

2.1. Modularity

While each model has its own specific properties (*e.g.,* decision tree in ProtoTree, scoring sheet in PIP-Net, linear layer in ProtoPNet), CBR image classifiers from the state of the art [1, 2, 3, 4, 5, 6] share a common architecture, displayed on Figure 1. In CaBRNet, *this common architecture can be parametrized at will* using a set of YAML files that are stored in each training directory for reproducibility (see Sec. 2.3).

More precisely, as shown in Figure 1 and Table 1, the architecture of a CBR image classifier starts with a backbone – usually the feature extractor of an existing convolutional neural network (CNN) such as ResNet50 – with optional additional layers to reduce dimensionality (*e.g.,* the fourth layer of a ResNet50 followed by a convolutional layer with sigmoid activation).

The feature extractor outputs a set of vectors, that form the latent space, and is followed by a similarity layer whose role is to compute the similarity between the latent vectors of the input image and a series of reference vectors, called *prototypes*, each obtained from an image from the training set. From the similarity scores, the process of locating the prototypes inside images (attribution) and visualizing the relevant pixels (*e.g.,* bounding box, cropping, heatmaps) is also parametrized by a configuration file, as described in Table 1. Finally, a decision
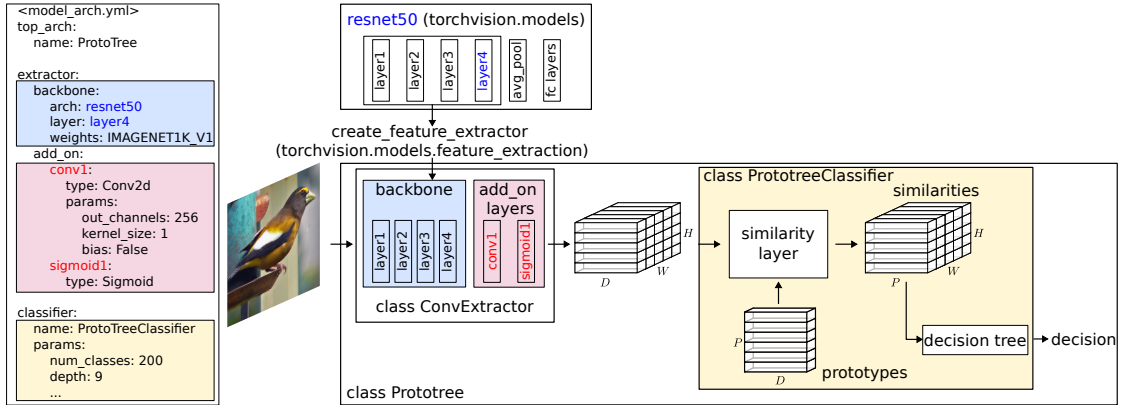
Figure 1: Striving for modularity. From a YAML configuration file (left), CaBRNet backend exploits the common architecture of CBR image classifiers to simplify the instantiation of new models (here, a ProtoTree).

layer, parametrized by an architecture-dependent python class (*e.g.,* decision tree, linear layer), assigns a score to each class based on the similarity scores, the highest being the decision (object classification).

In addition to these architectural choices, CaBRNet also allows specifying the parameters necessary for the training of the models (*e.g.,* objective functions, number of epochs, optimizer to use, which parts of the neural network can be updated and when, training dataset, preprocessing). For more information regarding the training and data configuration, we refer the reader to the CaBRNet documentation.

In its current version (v0.2), and as shown in Table 1, CaBRNet implements two architectures (ProtoPNet and ProtoTree) and five attribution methods:

- Upsampling of the similarity map with cubic interpolation, as in [1, 2, 4, 3, 5, 6];
- SmoothGrads[12]/backpropagation[13], as proposed in [10];
- PRP[9], a variant of LRP[14] with a propagation rule for the similarity layer;
- RandGrads, a dummy attribution method returning random gradients (used as a baseline when comparing attribution methods in [10]).

In the coming months, we plan to add support for more architectures (see Sec. 4).

2.2. Backward compatibility

Significant work has already been carried out on CBR classifiers. Therefore, ensuring that any result obtained with previously existing codebases (*e.g.,* model training) can be reused in the CaBRNet framework is paramount and has been one of our main and earliest priorities, in an effort to ensure backward compatibility.

First, *any previously trained model can be loaded and imported within the CaBRNet framework*, *i.e.,* models trained using the original code proposed by the authors of ProtoPNet and ProtoTree

Table 1

CaBRNet main configuration parameters (for a complete list, please refer to the documentation).

Parameter	Description	Supported values
Model configuration		
extractor/backbone/arch	Backbone of the extractor	Any from torchvision.models
extractor/backbone/layer	Where to extract features	Any layer inside backbone
extractor/add_on	Add-on layers of extractor	Any from torch.nn
classifier	CBR classifier	ProtoPNet or ProtoTree
Data configuration		
train_set/name	Training dataset class	Any from torchvision.datasets
train_set/params/transform	Data preprocessing	Any from torchvision.transforms
Visualization configuration		
attribution/type	Attribution method	Upsampling, SmoothGrad, Backprop, PRP, RandGrads
view/type	How to visualize patches	

can be used by our framework for other purposes (*e.g.,* benchmarks) without having to retrain the model from scratch.

Second, our implementation of existing architectures (currently, ProtoPNet and ProtoTree) supports two modes: i) default mode, where all operations have been reimplemented so that they are as up-to-date as possible with the latest PyTorch versions; ii) compatibility mode, as a sanity check (*i.e.,* to make sure that our implementation does not deviate from previous implementations), whose purpose is to be accurate with previous works at the operation level[1].

2.3. Reproducibility and transparency

Reproducibility is key for the transparency of research and good software engineering alike. However, replicating machine learning (ML) results can be particularly tricky[2]. One reason for this is the inherent randomness at the core of ML programs (and the lack of documentation of settings related to that randomness). The rapid update pace of common ML frameworks

---

[1]Backward compatibility was rigorously tested using unit tests covering all aspects of the process, from data loading to model training, to pruning and prototype projection.
[2]See https://reproducible.cs.princeton.edu/

leads to the regular deprecation of APIs and features, which may make a code impossible to run without directly changing its source, or going through the rabbit hole of dependency hell. As an example, the mixed-precision training setting, available from PyTorch v1.10, uses different types for CPU and GPU computations, which may lead to different results. One last part is the under-specification of the dataset constitution (curation process, classes imbalance, *etc.*) and preprocessing pipelines (test/train splits, data transformations).

Reproducibility can cover many notions. In [15], it is defined as the following: "A ML study is reproducible if readers can fully replicate the exact results reported in the paper." We strive to provide a similar definition: "the same set of parameters will always yield the same results", with the following limitations:

- reproducibility can be ensured only for a given hardware/software configuration. The software configuration is specified through the requirements.txt. We are aware that stricter software environment specifications do exist (for instance, stateless build systems like Nix[3]). Integrating such solutions into CaBRNet would be an interesting prospect, but also comes with an additional engineering cost to keep the library's ease of use. While not currently supported, we plan to save information regarding the hardware configuration that led to a given result;
- hyperparameters that seem innocuous may influence the results. For instance, the size of the data batches - **even during testing** - has a small influence on the results. Consequently, we also save this information inside configuration files.

To address the randomness variation, we initialize the various random number generators (RNGs) using a fixed seed that is stored along the training parameters. Thus, while loading a model's parameters, the seed used for its training will be loaded as well, guaranteeing that given the same hardware and hyperparameter configuration, the variability induced by the pseudo-randomness is identical. We also support the possibility to save checkpoints at various steps of the process, and we include the current state of all RNGs to restore the training process exactly as it was.

As we believe that reproducibility is improved by a good documentation, we also provide detailed installation instructions, a tutorial, the API reference and the CaBRNet backend reference. The major part of this documentation is automatically generated from the source code, ensuring consistency between the code and the documentation at all times.

3. ... and in the light evaluate them

In this section, we describe our design choices for the CaBRNet benchmark, with the purpose of proposing a framework for the systematic evaluation of CBR models.

3.1. Going beyond accuracy

Current CBR models rely on two assumptions: i) proximity in the latent space (*w.r.t.* a given distance metric) is equivalent to similarity in the visual space; ii) there exists a simple mapping
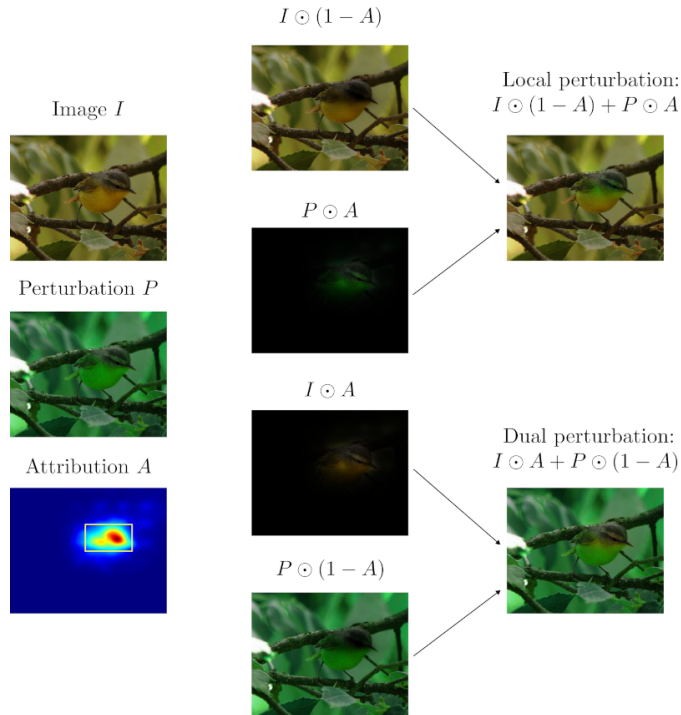
---

[3]https://nixos.org/

Figure 2: Improvement over the perturbation metric used in [7]. Rather than studying the drop in similarity score following a global perturbation of the image (*e.g.,* shift in the hue of the image), we apply a *local* perturbation using the heatmap produced by the chosen attribution technique. Hence, we not only measure the sensitivity of the similarity score to a given perturbation, but also the ability of the attribution method to locate the most relevant pixels. Additionally, we measure the drop in similarity when applying the perturbation to *anything but* the most important pixels (dual perturbation).

between a latent vector and a localized region in the original image, due to the architecture of the feature extractor (CNN). These assumptions have recently been put to the test by various metrics[4], that are already implemented or that we aim to implement in CaBRNet to streamline the evaluation of CBR models. In particular, CaBRNet currently integrates:

- the perturbation-based explanation of [7], with improvements: perturbations are no longer applied to the entire image, but to the identified patch of image corresponding to the prototypical part, as shown in Figure 2;
- the pointing game (relevance metric) of [10], with improvements: the metric now also supports the energy-based pointing game introduced in [16].

In the coming months, we plan to add support for more evaluation metrics (see Sec. 4).

3.2. Stop wasting time retraining state-of-the-art models

In our opinion, reproducing results from the state of the art (*i.e.,* re-training models) – to serve as points of comparison with a new approach – can waste valuable time and computing resources

---

[4]Regarding the relevance and/or effectiveness of these metrics, we refer the reader to the original papers.

from AI researchers, when that time could be dedicated to improving that new approach. This issue mostly arises when proposing new evaluation metrics[5] that must be applied to a wide range of state-of-the-art models. Thus, one of the objectives for CaBRNet is to *publish pre-trained state-of-the-art CBR models, so that they can be used readily by the XAI research community*. For the sake of transparency, and as stated in Sec. 2.3, each published model is also associated with information ensuring a level of reproducibility, should a researcher wish to re-train these models. Moreover, in an effort to improve the statistical significance of all related experiments, *we plan to publish at least three models per model configuration and dataset.* As an example, we have already published 6 models[6] trained on the Caltech-UCSD Birds 200 (CUB200 [17]) using our implementation of ProtoTree with trees with depth 9 and 10.

4. Conclusion and future works

CaBRNet is open-source and welcomes any external contribution. We also strongly encourage the research community to publish trained models that can be reused within our evaluation framework. On our side, future developments include: i) support for ProtoPool[3], ProtoPShare[4], PIP-Net[6] and TesNet[5]; ii) support for the metric measuring the stability to JPEG compression [8] and the stability to adversarial attacks[11]; iii) hands-on tutorials for gently introducing users to the various aspects of the framework.

Ideally, the modularity of CaBRNet design will appeal to both AI researchers wanting to experiment with CBR approaches, and industrial actors interested in deploying those approaches in a principled and traceable way. Our goal is for CaBRNet to become the development framework for new and innovative CBR approaches by the community.

Acknowledgments

---

[5]When using common metrics (*e.g.,* model accuracy), it is possible to avoid retraining models by simply referring to the results provided by the original authors.

[6]Available at https://zenodo.org/records/10894996.

References

[1] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, C. Rudin, *This* looks like *That*: Deep learning for interpretable image recognition, NeurIPS (2019) 8930–8941.

[2] M. Nauta, R. van Bree, C. Seifert, Neural prototype trees for interpretable fine-grained image recognition, CVPR (2021) 14928–14938.

[3] D. Rymarczyk, L. Struski, M. Górszczak, K. Lewandowska, J. Tabor, B. Zieliński, Interpretable image classification with differentiable prototypes assignment, in: ECCV, 2021.

[4] D. Rymarczyk, L. Struski, J. Tabor, B. Zieliński, ProtoPShare: Prototypical parts sharing for similarity discovery in interpretable image classification, SIGKDD (2021).

[5] J. Wang, H. Liu, X. Wang, L. Jing, Interpretable image recognition by constructing transparent embedding space, ICCV (2021) 875–884.

[6] M. Nauta, J. Schlötterer, M. van Keulen, C. Seifert, PIP-Net: Patch-based intuitive prototypes for interpretable image classification, in: CVPR, 2023, pp. 2744–2753.

[7] M. Nauta, A. Jutte, J. C. Provoost, C. Seifert, This looks like that, because ... explaining prototypes for interpretable image recognition, in: PKDD/ECML Workshops, 2020.

[8] A. Hoffmann, C. Fanconi, R. Rade, J. Kohler, This looks like that... does it? shortcomings of latent space prototype interpretability in deep networks, ICML Workshop on Theoretic Foundation, Criticism, and Application Trend of XAI (2021).

[9] S. Gautam, M. M.-C. Höhne, S. Hansen, R. Jenssen, M. Kampffmeyer, This looks more like that: Enhancing self-explaining models by prototypical relevance propagation, Pattern Recognition (2022) 109172.

[10] R. Xu-Darme, G. Quénot, Z. Chihani, M.-C. Rousset, Sanity checks for patch visualisation in prototype-based image classification, XAI4CV at CVPR (2023).

[11] M. Sacha, B. Jura, D. Rymarczyk, L. Struski, J. Tabor, B. Zieliński, Interpretability benchmark for evaluating spatial misalignment of prototypical parts explanations, ArXiv abs/2308.08162 (2023).

[12] D. Smilkov, N. Thorat, B. Kim, F. B. Viégas, M. Wattenberg, Smoothgrad: removing noise by adding noise, ICML Workshop on Visualization for Deep Learning (2017).

[13] K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: Visualising image classification models and saliency maps, Workshop at ICLR (2014).

[14] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, W. Samek, On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation, PLoS ONE 10 (2015).

[15] M. B. A. McDermott, S. Wang, N. Marinsek, R. Ranganath, L. Foschini, M. Ghassemi, Reproducibility in Machine Learning for health research: Still a ways to go, Science Translational Medicine 13 (2021).

[16] H. Wang, Z. Wang, M. Du, F. Yang, Z. Zhang, S. Ding, P. P. Mardziel, X. Hu, Score-CAM: Score-weighted visual explanations for convolutional neural networks, CVPRW (2019) 111–119.

[17] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, P. Perona, Caltech-UCSD Birds 200, Technical Report CNS-TR-2010-001, CalTech, 2010.