Software for Dataset-wide XAI: From Local Explanations to Global Insights with Zennit, CoRelAy, and ViRelAy

Preprint · June 2021

CITATIONS
0

READS
245

5 authors, including:

Wojciech Samek
Technische Universität Berlin
262 PUBLICATIONS **21,949** CITATIONS
SEE PROFILE

Klaus-Robert Müller
Technische Universität Berlin
955 PUBLICATIONS **101,550** CITATIONS
SEE PROFILE

Sebastian Lapuschkin
Fraunhofer-Institut für Nachrichtentechnik, Heinrich-Hertz-Institut
86 PUBLICATIONS **10,097** CITATIONS
SEE PROFILE

Software for Dataset-wide XAI: From Local Explanations to Global Insights with Zennit, CoRelAy, and ViRelAy

Christopher J. Anders[1,2], David Neumann[3], Wojciech Samek[2,3], Klaus-Robert Müller[1,2,4,5], and Sebastian Lapuschkin[3]

[1] *Machine Learning Group, Technische Universität Berlin, Berlin, Germany*
[2] *BIFOLD – Berlin Institute for the Foundations of Learning and Data, Berlin, Germany*
[3] *Department of Artificial Intelligence, Fraunhofer Heinrich Hertz Institute, Berlin, Germany*
[4] *Department of Artificial Intelligence, Korea University, Seoul, Korea*
[5] *Max Planck Institut für Informatik, Saarbrücken, Germany*

Abstract

Deep Neural Networks (DNNs) are known to be strong predictors, but their prediction strategies can rarely be understood. With recent advances in Explainable Artificial Intelligence, approaches are available to explore the reasoning behind those complex models' predictions. One class of approaches are post-hoc attribution methods, among which Layer-wise Relevance Propagation (LRP) shows high performance. However, the attempt at understanding a DNN's reasoning often stops at the attributions obtained for individual samples in input space, leaving the potential for deeper quantitative analyses untouched. As a manual analysis without the right tools is often unnecessarily labor intensive, we introduce three software packages targeted at scientists to explore model reasoning using attribution approaches and beyond: (1) *Zennit* – a highly customizable and intuitive attribution framework implementing LRP and related approaches in PyTorch, (2) *CoRelAy* – a framework to easily and quickly construct quantitative analysis pipelines for dataset-wide analyses of explanations, and (3) *ViRelAy* – a web-application to interactively explore data, attributions, and analysis results.

1 Introduction

There is no doubt that Deep Neural Networks (DNNs) are strong predictors, which have solved many problems far and wide [1–3]. With their inherent complexity, however, also comes a heavy down-side, which is the lack of transparency of DNNs. Recent advances in Explainable Artificial Intelligence (XAI) (see, *e.g.*, [4, 5] for a timely overview), however, allow for a more in-depth investigation of DNN behavior. Here, attribution methods are able to yield local explanations, *i.e.*, attribution scores for all (input) features of individual samples. The Layer-wise Relevance Propagation (LRP) [6, 7], for example, with its mathematical roots in Deep Taylor Decomposition (DTD) [8] and its various purposed modified backpropagation rules [7, 9, 10], has proven to be a particularly powerful method of local XAI showing excellent results [9, 11–13] when recommended guidelines are followed, yet it is rarely used *to its full potential*, *e.g.*, due to a lack of ready-made and *complete* implementations. In particular, an exhaustive implementation of LRP following contemporary recommendations from literature [5, 7, 9] is still lacking for the popular PyTorch framework. As one of our contributions, we thus aim to make a *proper and flexible* implementation of LRP available to the community, which goes beyond the simple variants LRP-ε or (Gradient×Input) often provided as the sole [14], and not universally recommended variants, of the method.

If employed correctly, local XAI has the potential to point out previously unknown but interesting model behavior, or biased and artifactual predictions [15, 16]. With very large datasets however, a thorough (manual) analysis of attribution results, *e.g.*, for the understanding and verification of model behavior, or the discovery of systematic misbehavior are very labor- and time-intensive. Still, further insight beyond local attributions is required, *e.g.*, to understand global model behavior, or to notice methodical Clever Hans [17, 18] traits of a model. Recent approaches such as Spectral Relevance Analysis (SpRAy) [18] provide a solution to this arduous task by automating large parts of the analysis workload and are thus, together with appropriate visualizations, aiding in the discovery of prediction strategies employed by a DNN model.

In this paper, we introduce a triad of software packages targeted at scientists to explore the reasoning of machine learning models based on dataset-wide XAI:

1. With *Zennit* we provide a highly customizable, yet intuitive local XAI framework, for PyTorch. It is focused on rule-based approaches such as LRP and based on PyTorch's `Module` structure, enabling (and delivering) implementations of various attribution methods.

2. *CoRelAy* in turn digests attributions (and possibly also other sources of data), and can be used to quickly build elaborate, dataset-wide analysis pipelines such as SpRAy, consisting of, *e.g.*, processing, clustering and embedding steps. The framework aims at efficiency during analysis by re-using matching (partial) pipeline results as often as possible within and between pipeline executions, instead of re-computing the complete pipeline each time, *e.g.*, due to parameter changes.

3. *ViRelAy* provides a user-friendly entry point to the analysis results from *Zennit* and *CoRelAy* in form of an interactive web-application. During the exploration of data with model attributions, clusterings, and visualizable embeddings, researchers can import, export, bookmark, and share particular findings with their peers.

In combination, these three tools enable XAI to be used to quantitatively and qualitatively explore and investigate large scale models and data: Local model explanations can be obtained through attributions computed with *Zennit*[1]. Users may then analyze large sets of attributions computed over whole datasets with pipelines built in *CoRelAy*[2], of which the results can then be visualized and investigated with *ViRelAy*[3]. The insights obtainable through this particular, yet flexible recipe allows to go beyond passively observant XAI, *e.g.*, by fuelling a strategy of informed intervention; only through the use of the here introduced scalable software packages, we were able to identify systematically biased reasoning in DNN models trained on ImageNet [19].

Related Work Multiple software frameworks have been introduced using different deep learning libraries to compute model attributions. One of the earlier and comprehensive XAI software packages is the LRP Toolbox [20], providing implementations of a wide array of recommended LRP decomposition rules for the Caffe Deep Learning Framework [21], as well as Matlab and Python (using NumPy [22] and CuPy [23]) via custom neural network interfaces. The software framework iNNvestigate [24], which is based on TensorFlow [25] and Keras [26], implements LRP and other attribution approaches. While it provides a straight-forward approach to apply multiple attribution methods on existing Keras models, its structure makes customization (*e.g.*, by implementing custom rules and compositions of rules) non-trivial. Captum [14], which is tightly integrated into PyTorch, provides a broad spectrum of attribution methods. It is very customizable, but lacks specificity for layer-type specific implementations of decomposition rules necessary for LRP, thus requiring a lot of work to use state-of-the-art defaults for LRP. TorchRay [27] is another attribution framework built on PyTorch, which also provides a broad spectrum of attribution methods, but has no support for LRP.

2 Attribution with Zennit

Zennit provides a framework for attribution in PyTorch [28]. It is based on the `Module` structure in PyTorch, and makes heavy use of its `Autograd` and `Hook` functionalities. It is mainly focused on implementing the rule-based approach used by LRP [6] in a simple and intuitive manner: The provision of an easy to modify and flexible implementation of LRP is paramount for obtaining excellent results, by optimally aligning the method to the characteristics of the model (or parts thereof) to be analyzed [5, 7, 9].

Simpler attribution methods, such as SmoothGrad [29] and Integrated Gradients [30], are also implemented, although they do not make use of the rule-based system, but are straight forward functions of the gradient of the model to be analyzed.

[1] `https://github.com/chr5tphr/zennit`
[2] `https://github.com/virelay/corelay`
[3] `https://github.com/virelay/virelay`

Rule-Based Attributions Rule-based attribution methods assign different rules to `Modules` within a model depending on the function and context. In *Zennit*, rule-based attributions are computed by attaching *forward* and *backward* `Hooks` to `Modules` (layers), such that computing the gradient of the model will instead provide the desired attribution. At the heart of *Zennit* is the `BasicHook`, which contains the functionality to register and remove modifications to a single `Module` (layer), and a general attribution method. *Rules* are created by providing functions to a `BasicHook` to customize the general attribution method with modified inputs, parameters, and accumulators. This makes an implementation of new rules trivial. All popular rules for LRP (for an overview see [7]), as well as others, such as GuidedBackprop [31] and ExcitationBackprop [32], come pre-implemented.

Mapping Rules with Composites The biggest problem when dealing with rule-based attribution methods is to assign the desired rules to all individual layers. *Zennit* solves this by implementing `Composites`, which are mappings from `Module`-properties to rules. `Module`-properties are for example the name or type of function, its (hyper-)parameters or its position within the model. `Composites` are provided with a `module_map`, which, given the `Module`-properties, returns a template-rule to be assigned to the layer. One example for a built-in basic `Composite` is the `SpecialFirstLayerMapComposite`, which assigns rules based on layer types, but handles the first linear layer differently. This is the basis for most LRP-based `Composites` for feed-forward networks, like `EpsilonGammaBox`, which uses the LRP-ε-rule for dense layers, the LRP-γ-rule for convolutional layers, and the LRP-Z^B-rule (or box-rule) for the first convolutional layer [7].

Temporary Model Modification with Canonizers Another problem with rule-based attribution methods is that their rules may not directly be applicable to many networks, unless they are transformed into a canonical form [5, 33, 34]. For example, multiple consecutive linear layers with only one activation at the very end cannot always be trivially attributed with all variants of LRP unless the consecutive linear layers are merged into a single one. A common example for this structure is batch normalization [35]. To temporarily modify models in-place into a canonical form, *Zennit* implements `Canonizers`. Due its common application, *Zennit* provides the `MergeBatchNorm Canonizer`, to temporarily merge batch normalization layers into an adjacent linear layer [36–38]. A general `Canonizer`, which is, for example, needed to apply LRP on ResNet [39, 40], is the `AttributeCanonizer`, which, while registered, will modify (instance) attributes in place, for example, to split a single module for which there is no rule, into multiple ones for which rules may then be assigned. Model-specific `Canonizers` for popular models like VGG-16 [41] and ResNet [39, 40] from, *e.g.*, Torchvision [42], are implemented for convenience. `Canonizers` are directly provided to `Composites`, so they will be applied right before the rules are mapped to the layers when registering the `Composite` to a model.

Attributors Attributors are optional convenience functions to either compute the gradient given a model and a `Composite`, or to implement black-box attribution approaches such as SmoothGrad and Integrated Gradients. Given a gradient-based black-box attribution approach, *e.g.*, SmoothGrad, it is also possible to supply a `Composite`, to compute a combination of, *e.g.*, LRP and SmoothGrad, since the composite will modify the gradient of the model. Non-gradient based approaches, like Occlusion Analysis [43], cannot be combined with `Composites`, since the modified gradient of the `Composite` has no effect on the result.

Heatmaps Since attributions for image data are often visualized in heatmaps, *Zennit* comes with an image module to easily visualize and store attributions as heatmap images. Various color maps are available. The images are stored using intensities and 8-bit palettes where indices correspond to the attributed relevances. This makes it easy to change the color map afterwards, without re-computing the relevance values. An example for visualized heatmaps is given in Figure 1.

3 Building Analysis Pipelines with CoRelAy

While attribution methods can give a qualitative insight into a model's prediction strategies, a user may only guess how the attributions of individual heatmaps are part of the model's reasoning. A deeper insight into the model may be gained by conducting a dataset-wide analysis. Lapuschkin

Figure 1: Heatmaps of attributions of lighthouses, using the pre-trained VGG-16 network provided by Torchvision. The `Composite` `EpsilonGammaBox` was used and the attributions were visualized with the color map `coldnhot` (negative relevance is light-/blue, irrelevant pixels are black, positive relevance is red to yellow).

et al. [18] introduced Spectral Relevance Analysis, with which they quantitatively analyze a model's prediction strategy by visually embedding and clustering attributions with Spectral Clustering [44, 45] and t-distributed Stochastic Neighborhood Embedding (t-SNE) [46]. Anders et al. [19] extended SpRAy by using different clustering and visual embeddings, as well as computing a pre-ranking of interesting classes based on the linear separability of their clusterings. *CoRelAy* is a tool to quickly compose quantitative analysis pipelines like SpRAy, which provide multiple embeddings, representations, and labels of the data. While our main use-case and motivation for *CoRelAy* was to analyze attributions provided by *Zennit*, *CoRelAy* is not limited to any kind of data, *e.g.*, *CoRelAy* may also be used for a quick dataset exploration with multiple clusterings and embeddings.

Processors and Params `Processors` are the actions in a pipeline. To implement a `Processor`, an inheriting class will have to implement a method with the name `function`, and class-scope `Params`. In Python terminology, `Params` are descriptors, which change based on the instance they are bound to (similar to methods). `Params` are used to easily define the arguments of `Processors`, their desired types, default values, and others. `Processors` already have the `Params` `is_output`, to signal that the output of this `Processor` should be returned by the `Pipeline` (even if intermediate), and `io`, which can be assigned to a `Storage` object to cache data on disk. Many `Processors` come pre-implemented with *CoRelAy*, which are categorized into pre-processing, distance functions, affinity functions, Laplacians, embedding methods, and flow `Processors`. Flow `Processors` are used to design more complex flows of `Pipelines`, of which the most important are `Parallel` and `Sequential`. With `Parallel`, the output of the previous `Processor` may be passed to multiple `Processors`, *e.g.*, to compute multiple clusterings on the same data or to try to compute a visual embedding with different hyperparameters. With `Sequential`, `Processors` may be combined to do multiple steps where there is only a single `Task` in a `Pipeline`.

```python
with h5py.File('spray.h5', 'a') as fd:
    iobj = HashedHDF5(fd.require_group('proc_data'))
    pipeline = SpectralClustering(
        embedding=EigenDecomposition(n_eigval=8, io=iobj),
        clustering=Parallel([
            Parallel([
                KMeans(n_clusters=k, io=iobj) for k in range(2, 20)
            ], broadcast=True),
            TSNEEmbedding(io=iobj)
        ], broadcast=True, is_output=True)
    )
    data = numpy.random.normal(size=(64, 3, 32, 32))
    clusterings, tsne = pipeline(data)
```

Figure 2: Example code to instantiate and execute a simple SpRAy pipeline, using 8 eigenvalues for the Spectral Embedding, clustering using k-means with $k \in \{2, ..., 20\}$, and visualizing with t-SNE. The results are additionally cached in a file called `spray.h5`.

Pipelines and Tasks `Pipelines` are feed-forward functions, which have `Tasks` that have to be fulfilled from front to back to execute the pipeline. In *CoRelAy*, `Pipelines` can be seen as computation templates, where there are steps involved to compute a certain result, which can be individually changed. A `Task` is such a step, with a default `Processor`, and optionally an allowed type of `Processor`. When instantiating a `Pipeline`, `Tasks` may be assigned a new `Processor` to handle the data instead of the default one. A `Pipeline` can be executed by simply calling it as a function with the input data as its arguments. Depending on the `Processors` used and their respective `is_output` flags, the output of the `Pipeline` may have none, one, or a hierarchy of results. If `Processors` within the `Pipeline` own an `io` object, they will cache their results by hashing the input data and parameters. When calling the same `Pipeline` with the same data, these results will be looked up instead of being re-computed. *CoRelAy* has a SpRAy `Pipeline` (*cf.* [19]) pre-implemented, to produce data which can be directly used with *ViRelAy*. A SpRAy `Pipeline` may be instantiated and executed as shown in Figure 2.

4 Interactive Visualization with ViRelAy

With quantitative analyses, a large amount of results are created, and it may become hard to connect the different results and representations with the original data. A labor-intensive manual comparison and creation of individual plots, in an attempt to extract the essence of the results may become inevitable to find correlations in the data. The analysis performed with SpRAy has a very distinct and common set of objects that need to be compared: the source data points, their attributions (wrt. a model), a visual 2-dimensional representation of the (embedded) attribution data, clustering labels and global auxiliary scores. *ViRelAy* is an interactive web-application, with which the results may be freely explored by visually connecting these 5 objects. *ViRelAy*'s back-end is implemented in Python using Flask [47], and its front-end is implemented using Angular [48].

Data Loading *ViRelAy* is designed to process the data of *CoRelAy*. The results of *CoRelAy* are stored in HDF5 [49] files in a hierarchy that *ViRelAy* is able to use post-hoc, reducing loading times for an improved user interaction quality. The analysis file, along with the source data and the attribution data, both also stored in HDF5, are referenced in a project file. A single project file may contain one source dataset with one attribution for each sample, as well as an arbitrary amount of analysis files. To compare different datasets or attribution approaches, *ViRelAy* can be executed by supplying an arbitrary amount of project files, between which the client may switch during execution.

Explorative User Interaction The user interface is shown in Figure 3. At the top of the interface is (1) the project selection, where the projects, as dictated by the project files, show up as tabs and may be selected to switch between datasets and attribution methods. Below the project selection, on the left side is (2) the analysis selection, where the analysis approach (given by supplying multiple analysis files in a single project file), the category (which often is the data label, but may be chosen as any group of data points), the clustering method (which influences (8) the available clusters and (6) the data point coloring), and the embedding (which is the 2d representation of the data points as shown in (6) the visualization canvas) can be selected. Selecting a different analysis method resets all categories. To the right is (3) the color map selection, which changes the color map used in (9) the data/attribution selection, with a color bar indicating low (left) and high (right) values. The next item to the right is (4) the data/attribution visualization mode selection, which changes whether (9) the data/attribution visualization shows the source data (input), its attribution with the selected color map (attribution), or the attribution superimposed onto a gray-scale image of the source data (overlay). The (5) *import* and *export* buttons allow to export the currently selected analysis, category, clustering, embedding, color map, visualization mode and selected points by downloading a JSON-file, or importing a JSON-file to change the selections to the configuration of a previously downloaded file. This may be used either to remember or to share interesting results. The selection may also be shared or bookmarked in the form of a URL using the (5) *share* button. At the center of the interface is (6) the 2d-visualization canvas, which shows the points in the selected 2-dimensional embedding space (produced by, *e.g.*, t-SNE) colored by the clusters indicated in (8) the cluster point selection. In this canvas, the user may zoom or pan, and select points which will be highlighted by a more saturated color and shown in

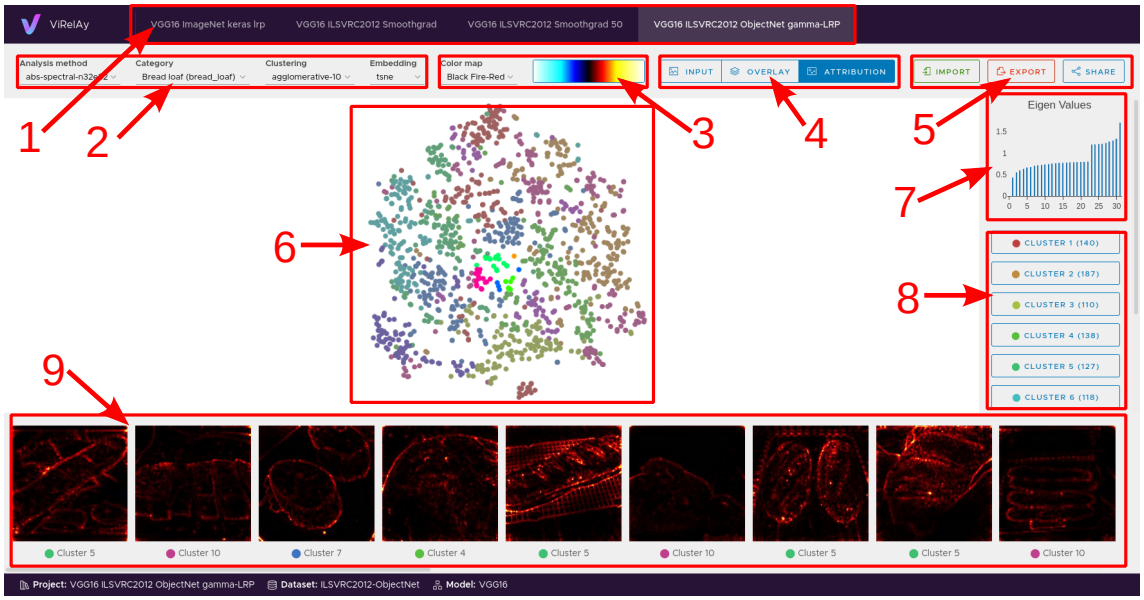

Figure 3: The *ViRelAy* user interface. Highlighted points are: (1) – Project selection, (2) – analysis setup and category selection, (3) – color map selection, (4) – data/attribution visualization mode selection, (5) – import/export/share current selection, (6) – 2d visual embedding canvas, (7) – auxiliary score plot, (8) – cluster point selection, (9) – data/attribution visualization.

(9) the data/attribution visualization. Hovering over data points will show a preview of the source data inside the canvas. To the right is (7) the auxiliary category score plot, which in Figure 3 are the eigenvalues of the Spectral Embedding. Below, there is (8) the cluster point selection, which shows the available clusters of the selected clustering, as well as the colors used for members of these clusters in (6) the 2d-visualization canvas, and the number of points in this cluster in parentheses. Finally, at the bottom is (9) the data/attribution visualization, where, depending on which mode was selected in (4) the data/attribution mode selection, will show either the source data, the attribution heatmap, or the attribution superimposed on a gray-scale version of the source image, of a subset of the selected points.

5 Conclusion

In advocacy of reproducibility in machine learning [50], we have introduced three open source software frameworks to attribute, analyze, and interactively explore a model's dataset-wide prediction strategies: With *Zennit*, we hope to provide an intuitive tool within the boundaries of PyTorch to compute attributions in a customizable and intuitive fashion, and to make the multitude of rules in LRP and other rule-based attribution methods more accessible. We especially hope that any kind of model can now be analyzed by extending attribution approaches easily based on the intuitive structure of *Zennit*. By introducing *CoRelAy*, we hope to provide a simple way to analyze attributions dataset-wide in swiftly built pipelines, and thus explore the unused potential of insight into prediction models. Using *ViRelAy*, we hope to make the exploration of analysis results as effortless as possible by providing an interactive combined viewer of source data, attributions, visual embeddings, clusterings, and others. *Zennit*, *CoRelAy*, and *ViRelAy* in combination have already been successfully used in the analysis of ImageNet [51] on millions of images to find artifactual Clever Hans behavior [19], thus demonstrating effectiveness and scalability. With the frameworks' introduction, we hope to aid the community in the research and application of methods of XAI and beyond, to gain deeper insights into the prediction strategies of DNNs.

Acknowledgements

This work was supported in part by the German Ministry for Education and Research (BMBF) under grants 01IS14013A-E, 01GQ1115, 01GQ0850, 01IS18056A, 01IS18025A and 01IS18037A. This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 965221, and is also supported by the Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (No. 2017-0-001779), as well as by the Research Training Group "Differential Equation- and Data-driven Models in Life Sciences and Fluid Dynamics (DAEDALUS)" (GRK 2433) and Grant Math+, EXC 2046/1, Project ID 390685689 both funded by the German Research Foundation (DFG).

References

[1] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[2] K. T. Schütt, F. Arbabzadah, S. Chmiela, K.-R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nature Communications*, vol. 8, p. 13890, 2017.

[3] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Zídek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu, and D. Hassabis, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[4] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Computing Surveys*, vol. 51, no. 5, pp. 93:1–93:42, 2019.

[5] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining deep neural networks and beyond: A review of methods and applications," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021.

[6] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, no. 7, p. e0130140, 2015.

[7] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, "Layer-wise relevance propagation: an overview," in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer LNCS 11700, 2019, pp. 193–209.

[8] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognition*, vol. 65, pp. 211–222, 2017.

[9] M. Kohlbrenner, A. Bauer, S. Nakajima, A. Binder, W. Samek, and S. Lapuschkin, "Towards best practice in explaining neural network decisions with lrp," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7.

[10] L. Arras, J. A. Arjona-Medina, M. Widrich, G. Montavon, M. Gillhofer, K.-R. Müller, S. Hochreiter, and W. Samek, "Explaining and interpreting lstms," in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, ser. Lecture Notes in Computer Science. Springer, 2019, vol. 11700, pp. 211–238.

[11] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Müller, "Evaluating the visualization of what a deep neural network has learned," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 11, pp. 2660–2673, 2017.

[12] N. Pörner, H. Schütze, and B. Roth, "Evaluating neural network explanation methods using hybrid documents and morphosyntactic agreement," in *Proceedings of the Association for Computational Linguistics, (ACL)*. Association for Computational Linguistics, 2018, pp. 340–350.

[13] L. Arras, A. Osman, and W. Samek, "Ground truth evaluation of neural network explanations with clevr-xai," *CoRR*, vol. abs/2003.07258, 2020.

[14] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan, and O. Reblitz-Richardson, "Captum: A unified and generic model interpretability library for pytorch," *CoRR*, vol. abs/2009.07896, 2020.

[15] S. Lapuschkin, A. Binder, G. Montavon, K.-R. Müller, and W. Samek, "Analyzing classifiers: Fisher vectors and deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2912–2920.

[16] J. Aeles, F. Horst, S. Lapuschkin, L. Lacourpaille, and F. Hug, "Revealing the unique features of each individual's muscle activation signatures," *Journal of the Royal Society Interface*, vol. 18, no. 174, p. 20200770, 2021.

[17] O. Pfungst, *Clever Hans: (the horse of Mr. Von Osten.) a contribution to experimental animal and human psychology*. Holt, Rinehart and Winston, 1911.

[18] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller, "Unmasking clever hans predictors and assessing what machines really learn," *Nature Communications*, vol. 10, p. 1096, 2019.

[19] C. J. Anders, L. Weber, D. Neumann, W. Samek, K.-R. Müller, and S. Lapuschkin, "Finding and removing clever hans: Using explanation methods to debug and improve deep models," *CoRR*, vol. abs/1912.11425, 2020.

[20] S. Lapuschkin, A. Binder, G. Montavon, K.-R. Müller, and W. Samek, "The LRP toolbox for artificial neural networks," *Journal of Machine Learning Research*, vol. 17, pp. 114:1–114:5, 2016.

[21] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[22] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020.

[23] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "Cupy: A numpy-compatible library for nvidia gpu calculations," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[24] M. Alber, S. Lapuschkin, P. Seegerer, M. Hägele, K. T. Schütt, G. Montavon, W. Samek, K.-R. Müller, S. Dähne, and P.-J. Kindermans, "iNNvestigate Neural Networks!" *Journal of Machine Learning Research*, vol. 20, pp. 93:1–93:8, 2019.

[25] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in {*USENIX*} *Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[26] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[27] R. Fong, M. Patrick, and A. Vedaldi, "Understanding deep networks via extremal perturbations and smooth masks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 2950–2958.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, 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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 8024–8035.

[29] D. Smilkov, N. Thorat, B. Kim, F. B. Viégas, and M. Wattenberg, "Smoothgrad: removing noise by adding noise," *CoRR*, vol. abs/1706.03825, 2017.

[30] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 3319–3328.

[31] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," in *Proceedings of the International Conference of Learning Representations (ICLR)*, 2015.

[32] J. Zhang, S. A. Bargal, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff, "Top-down neural attention by excitation backprop," *International Journal of Computer Vision*, vol. 126, no. 10, pp. 1084–1102, 2018.

[33] S. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K.-R. Müller, and W. Samek, "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognition*, vol. 115, p. 107899, 2021.

[34] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 756–795, 2021.

[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning, (ICML)*, ser. JMLR Workshop and Conference Proceedings, vol. 37. JMLR.org, 2015, pp. 448–456.

[36] L. Y. W. Hui and A. Binder, "Batchnorm decomposition for deep neural network interpretation," in *Proceedings of the International Work-Conference on Artificial Neural Networks (IWANN)*, ser. Lecture Notes in Computer Science, vol. 11507. Springer, 2019, pp. 280–291.

[37] M. Alber, "Efficient learning machines: From kernel methods to deep learning," Ph.D. dissertation, Technical University of Berlin, Germany, 2019.

[38] M. Guillemot, C. Heusele, R. Korichi, S. Schnebert, and L. Chen, "Breaking batch normalization for better explainability of deep neural networks through layer-wise relevance propagation," *CoRR*, vol. abs/2002.11018, 2020.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2016, pp. 770–778.

[40] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2016.

[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[42] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proceedings of the International Conference on Multimedia (ACM Multimedia)*. ACM, 2010, pp. 1485–1488.

[43] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[44] M. Meila and J. Shi, "A random walks view of spectral segmentation," in *Proceedings of the International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2001.

[45] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems*, 2002, pp. 849–856.

[46] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[47] M. Grinberg, *Flask Web Development - Developing Web Applications with Python*. O'Reilly, 2014.

[48] N. Jain, A. Bhansali, and D. Mehta, "Angularjs: A modern mvc framework in javascript," *Journal of Global Research in Computer Science*, vol. 5, no. 12, pp. 17–23, 2014.

[49] B. Fortner, "Hdf: The hierarchical data format," *Dr Dobb's J Software Tools Prof Program*, vol. 23, no. 5, p. 42, 1998.

[50] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. Müller, F. Pereira, C. E. Rasmussen, G. Rätsch, B. Schölkopf, A. J. Smola, P. Vincent, J. Weston, and R. C. Williamson, "The need for open source software in machine learning," *Journal of Machine Learning Research*, vol. 8, pp. 2443–2466, 2007.

[51] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.