How to iNNvestigate neural network’s predictions!

Maximilian Alber*, Sebastian Lapuschkin†, Philipp Seegerer*, Miriam Hägele*, Kristof T. Schütt‡*, Grégoire Montavon‡, Wojciech Samek†, Philipp Seegerer*, Miriam Hägele*, Kristof T. Schütt‡*, Sven Dähne*, Pieter-Jan Kindermans*
maximilian.alber@tu-berlin.de

*Technische Universität Berlin, 10623 Berlin, Germany
Machine Learning Group

†Fraunhofer Heinrich Hertz Institute, 10587 Berlin, Germany
Video Coding and Analytics

‡Korea University, Seoul 02841, Korea
Department of Brain and Cognitive Engineering

¶Max Planck Institute for Informatics, 66123 Saarbrücken, Germany

Abstract

In recent years, deep neural networks have revolutionized many application domains of machine learning and are key components of many critical decision or predictive processes such as autonomous driving or medical image analysis. In these and many other domains it is crucial that specialists can understand and analyze actions and predictions, even of the most complex neural network architectures. Despite these arguments neural networks are often treated as black boxes and their complex internal workings as well as the basis for their predictions are not fully understood.

In the attempt to alleviate this shortcoming many analysis methods were proposed, yet the lack of reference implementations often makes a systematic comparison between the methods a major effort. In this tutorial we present the library iNNvestigate which addresses the mentioned issue by providing a common interface and out-of-the-box implementation for many analysis methods. In the first part we will show how iNNvestigate enables users to easily compare such methods for neural networks. The second part will demonstrate how the underlying API abstracts common operations in neural network analysis and show how users can use them for the development of (future) methods.

iNNvestigate and the tutorial resources are available at:
https://github.com/albermax/innvestigate

1 Introduction

In recent years deep neural networks have revolutionized many domains, e.g., image recognition, speech recognition, speech synthesis, and knowledge discovery [Krizhevsky et al., 2012, LeCun et al., 2012, Schmidhuber, 2015, LeCun et al., 2015, Van Den Oord et al., 2016]. Due to their capabilities neural networks are already and will be widely used, i.a., to create compact knowledge representations, for knowledge discovery techniques and for critical decisions processes. Thus in

NIPS 2018 - MLOSS workshop.
applications like, e.g., comparative studies [Alber et al., 2017], in automatic learning [Zoph et al., 2017, Alber et al., 2018a] or chemical compound searches [Montavon et al., 2013, Schütt et al., 2017], it would be extremely useful to know which properties help a neural network to choose appropriate candidates. To fully leverage this potential it is essential that users can comprehend and analyze these processes.

Despite these arguments neural networks are often treated as black boxes, because their complex internal workings and the basis for their predictions are not fully understood. In the attempt to alleviate this shortcoming several methods were proposed, e.g., Saliency Map [Baehrens et al., 2010, Simonyan et al., 2013], SmoothGrad [Smilkov et al., 2017], IntegratedGradients [Sundararajan et al., 2017], Deconvnet [Zeiler and Fergus, 2014], GuidedBackprop [Springenberg et al., 2015], PatternNet and PatternAttribution [Kindermans et al., 2018], LRP [Bach et al., 2015, Lapuschkin et al., 2016a,b, Montavon et al., 2018], and DeepTaylor [Montavon et al., 2017]. Theoretically it is not clear which method solves the stated problems best, therefore an empirical comparison is required [Samek et al., 2017, Kindermans et al., 2017].

In this tutorial we present the library INNvestigate [Alber et al., 2018b] which provides a common interface to a variety of analysis methods and abstractions that enable fast and clean development of such methods. In particular, INNvestigate contributes:

- A common interface for a growing number of analysis methods that is applicable to a broad class of neural networks. With this instantiating a method is as uncomplicated as passing a trained neural network to it and allows for easy qualitative comparisons of methods. For quantitative evaluations of (image) classification task we further provide an implementation of the method “perturbation analysis” [Samek et al., 2017].

- Support of all methods listed above—this includes the first reference implementation for PatternNet and PatternAttribution and an extended implementation for LRP—and an open source repository for further contributions.

- A clean and modular implementation, casting each analysis in terms of layer-wise forward and backward computations. This limits code redundancy, takes advantage of automatic differentiation, and eases future integration of new methods.

The tutorial itself is composed of two parts:

- The first part focuses on the application of INNvestigate and will show how users can compare different analysis methods (for a single network) as well as how users can compare the prediction analyses of different neural networks (for a single method).

- The second part introduces the API of INNvestigate. This will be done in a step-by-step implementation of several analysis methods using the provided abstractions. This will facilitate users to extend and develop such methods with help of INNvestigate.

The remainder of this paper will outline and describe the library in more detail, while the resources for this tutorial are available at the project’s repository as Jupyter notebooks: https://github.com/albermax/innvestigate.

This manuscript is based on the following publication: Alber et al. [2018b].

2 Library

Interface  The main feature is a common interface to several analysis methods. The workflow is as simple as passing a Keras neural network model to instantiate an analyzer object for a desired algorithm. Then, if needed, the analyzer will be fitted to the data and eventually be used to analyze the model’s predictions. The corresponding Python code is:

```python
import innvestigate
model = create_a_keras_model()
analyzer = innvestigate.create_analyzer("analyzer_name", model)
analyzer.fit(X_train)  # if needed
analysis = analyzer.analyze(X_test)
```
Implemented methods  At publication time the following algorithms are supported: Gradient Saliency Map, SmoothGrad, IntegratedGradients, Deconvnet, GuidedBackprop, PatternNet and PatternAttribution, DeepTaylor, and LRP including LRP-Z, -Epsilon, -AlphaBeta. In contrast, current related work [Kotikalapudi et al., 2017, Ancona et al., 2018] is limited to gradient-based methods. We intend to further extend this selection and invite the community to contribute implementations as new methods emerge.

Documentation  The library’s documentation contains several introductory scripts and example applications. We demonstrate how the analyses can be applied to the following state-of-the-art models: VGG16 and VGG19 [Simonyan and Zisserman, 2014], InceptionV3 [Szegedy et al., 2016], ResNet50 [He et al., 2016], InceptionResNetV2 [Szegedy et al., 2017], DenseNet [Huang et al., 2017], NASNet mobile, and NASNet large [Zoph et al., 2017]. Figure 1 shows the result of each analysis on a subset of these networks.

![Figure 1: Result of methods applied to various neural networks (blank, if not applicable).](image)

2.1 Details

Modular implementation  All of the methods have in common that they perform a backpropagation from the model outputs to the inputs. The core of iNNvestigate is a set of base classes and functions that is designed to allow for rapid and easy development of such algorithms. The developer only needs to implement specific changes to the base algorithm and the library will take care of the complex and error-prone handling of the propagation along the graph structure. Further details can be found in the repositories documentation.

Training  PatternNet and PatternAttribution [Kindermans et al., 2018] are two novel approaches that condition their analysis on the data distribution. This is done by identifying the signal and noise direction for each neuron of a neural network. Our software scales favorably, e.g., one can train required patterns for the methods on large datasets like Imagenet [Deng et al., 2009] in less than an hour using one GPU. We present the first reference implementation of these methods.

Quantitative evaluation  Often analysis methods for neural networks are compared by qualitative (visual) inspection of the result. This can lead to subjective evaluations and one approach to create a more objective and quantitative comparison of analysis algorithms is the method “perturbation analysis” [Samek et al., 2017, also known as “PixelFlipping”]. The intuition behind this method is that perturbing regions which are recognized as important for the classification task by the analyzing method, will impact the classification most. This allows to assess which analysis method best identifies regions that matter for a specific task and neural network. iNNvestigate contains an implementation of this method.

Installation & license  iNNvestigate is published as open-source software and can be downloaded from: https://github.com/albermax/innvestigate. It is built as a Python 2 or 3 application on top of the popular and established Keras [Chollet et al., 2015] framework. The library can be simply installed as Python package.
Acknowledgments

This work was supported by the Federal Ministry of Education and Research (BMBF) for the Berlin Big Data Center BBDC (01IS14013A). This research was supported by Institute for Information & Communications Technology Promotion and funded by the Korea government (MSIT) (No. 2017-0-00451, No. 2017-0-01779).

References


François Chollet et al. Keras. https://keras.io, 2015.


