Latent Inspector: An Interactive Tool for Probing Neural Network Behaviors Through Arbitrary Latent Activation

Daniel Geißler , **Bo Zhou** and **Paul Lukowicz**
German Research Center for Artificial Intelligence (DFKI)
RPTU Kaiserslautern-Landau
{Daniel.Geissler, Bo.Zhou, Paul.Lukowicz}@dfki.de

Abstract

This work presents an active software instrument allowing deep learning architects to interactively inspect neural network models' output behavior from user-manipulated values in any latent layer. Latent Inspector offers multiple dimension reduction techniques to visualize the model's high dimensional latent layer output in human-perceptible, two-dimensional plots. The system is implemented with Node.js front end for interactive user input and Python back end for interacting with the model. By utilizing a general and modular architecture, our proposed solution dynamically adapts to a versatile range of models and data structures. Compared to already existing tools, our asynchronous approach of separating the training process from the inspection offers additional possibilities, such as interactive data generation, by actively working with the model instead of visualizing training logs. Overall, Latent Inspector demonstrates the possibilities as well as the appearing limits for providing a generalized, tool-based concept for enhancing model insight in terms of explainable and transparent AI.

1 Introduction

The steadily increasing amount of available data and the accompanying rise in model complexity makes it ever more challenging to understand and interpret the behaviors of machine learning models. Apart from the necessary design and training of machine learning models, verification and validation should not be neglected in the development workflow to ensure the trained models behave as the architects have intended. The intrinsic model property to behave like a black box, where input data is supplied and a specific output is given as result, requires proper methods to open the model in a way that a user can thoroughly probe the inner details. [Mohseni *et al.*, 2021; Samek *et al.*, 2017]

While there are other tools revealing the explainability and transparency such as CAM [Jung and Oh, 2021] or Tensorboard [Abadi *et al.*, 2015], there lacks a tool to trigger post-training models with arbitrary input or latent activation values, especially those outside the scope of the training data. In real-world applications, behaviors of those triggers can be important as real-world data can be unexpected, with faulty data sources, or with external factors that might interfere with the data integrity of the computing devices [Ziegler, 1996].

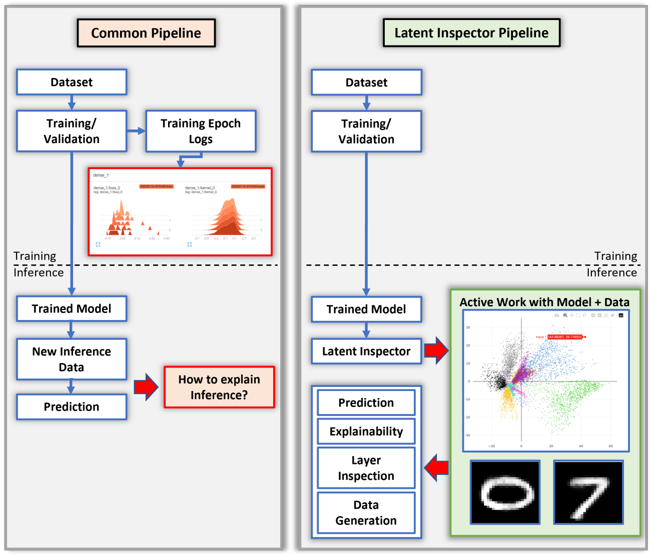

Figure 1: Common pipeline for training and inference of machine learning models compared to the approach of Latent Inspector

We introduce Latent Inspector, an interactive tool to circumvent the black box approach by disassembling the model into its fundamental parts without impairing functionality. Figure 1 shows the general pipeline for developing and analyzing Neural Networks in comparison to our adapted Latent Inspector pipeline. Instead of retrieving the necessary information in form of logging during the training process, our approach starts post-training as an inference intermediary. The tool dynamically adapts and actively works with the trained model by extracting, modifying, and processing the model's layer-based architecture and the available data. We provide several concepts to visualize the high-dimensional data processing in human-perceptible, two-dimensional plots. Furthermore, the user has the option to directly interact with the model through the tool, which enables extended functionalities like out-of-bounds inspection, fine-tuning layer weights, and data generation.

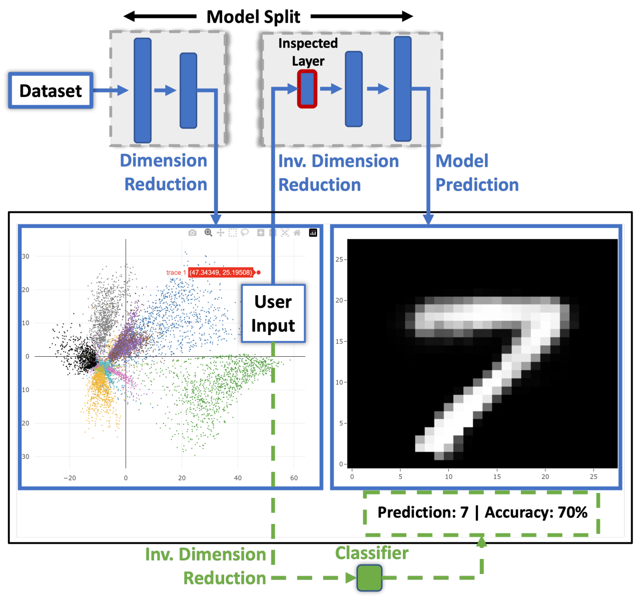

Figure 2: Latent Inspector's basic user interface with annotations to emphasize interaction with model and dataset

2 Functionality and Use Case

To offer potential users the possibility to probe their trained model in the Latent Inspector, we designed the tool as an active framework with a focus on generality and versatility. Together with a straightforward user interface and a plain file structure, most of the steps that are typically necessary to examine a custom model in this scope were transferred from the user to automatic and dynamic adapting functions inside the tool. In the following, implemented functionalities and potential use cases are introduced together with figure 2.

2.1 Functionality

The user interface follows a common and intuitive design layout containing fundamental controls in the sidebar and a grid system in the main window for interactive visualizations. After a short and simple setup process in which the user adds the necessary files to the designated folder in form of a new project, the Latent Inspector can directly be launched due to its ability of dynamically adapting to the provided files.

Dissecting the neural network model and accessing the architecture is a key element of our tool to realize the idea of inspecting every latent layer individually. The tool creates a list of the present layers from which the user may arbitrarily select one for inspection. An iterative process for retrieving layer properties allows the creation of sub-models based on the user selection. As described in figure 2, the model is split into two parts, one containing the preceding layers of the selection and the other containing the remaining part of the model. To maintain the sub-models fundamental operation, each layer is cloned from the original network to preserve the available parameters.

The user-provided data set is propagated through the first sub-network to obtain the input shape of the inspected latent layer. Usually, this preliminary output is further propagated to receive the final prediction, whereas, for the Latent Inspector, this output is utilized for visualizing the outcome as the latent space in a scatter plot on the left side of the main screen. If the user provides additional ground truth data within the project, each dot of the plot representing a data point is additionally colorized to highlight data distribution. The tool provides common unsupervised dimension reduction techniques like PCA, UMAP, or TSNE from the sklearn library to decrease the high dimensional output to displayable two dimensions [Pedregosa *et al.*, 2011]. Due to our universal approach, each dimension reduction technology aims to provide the best data representation by self-optimizing its parameters, considering the accompanying loss of information when transforming into lower dimensions.

To realize our novel feature to interactively work with the network, the latent space plot is designed to handle user click events. If the selected dimension reduction algorithm supports an inverse transformation, for instance, PCA and UMAP, the coordinates of the created data point are reversely scaled up into the inspected layer dimension and propagated through the second sub-network containing the remaining layers. The outcome of this process constitutes the model prediction of the user click event, which can be visualized through several predefined or customized components.

2.2 Use Case

A major difference between Latent Inspector and existing latent space visualizers is that our tool hosts the trained model and activates it with user-selected embedding values. Thus our tool is an active framework, which others like Tensorboard using only log file visualizations cannot achieve [Abadi *et al.*, 2015]. The user can trigger the network's output from any value at any latent layer through the interactive UI, resulting in the ability for out-of-bounds inspection to verify unknown or even impossible data points of real-world applications. Furthermore, we provide the option to save the propagated data points to files for the creation of new data sets or the refinement of existing ones.

To constitute the full potential of Latent Inspector, we demonstrate the functionality of data generation with user interactions in the case of classification based on an autoencoder feature extractor from the MNIST dataset [Bank *et al.*, 2020; Deng, 2012]. As emphasized with the green color in figure 2, our tool provides an extended feature to train a Logistic Regression classifier based on the inspected layer. The user needs to provide enough training and validation data beforehand to ensure proper classification. Again, our tool automatically adapts to the provided dataset with the help of the sklearn library, allowing either binary or multi-class classification [Pedregosa *et al.*, 2011]. For the MNIST use case scenario, the user input of the click event is propagated through the autoencoder to reconstruct the image of a handwritten number as well as through the trained classifier to label the autoencoder's output. Further, the user may manually modify the classifier label output via a text box if the prediction is incorrect, or to add new labels like for instance "not a number". The final result is stored in the tools project folder as a labeled dataset.

Additionally to the MNIST project, we added the PAMAP

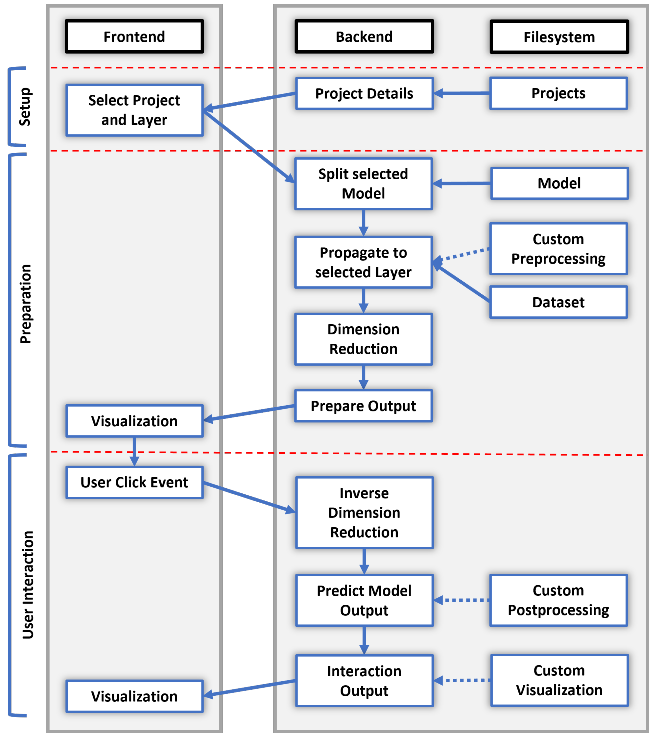

Figure 3: System Structure of Latent Inspector based on general front-end and back-end architecture

[Reiss and Stricker, 2012] and MoCaPose [Zhou *et al.*, 2023] projects as examples to validate our generalized and versatile approach. Apart from the introduced features, Latent Inspector is designed to be expanded with more functionalities due to its modular architecture.

3 System Structure

To achieve the above-mentioned functionalities and to keep Latent Inspector open for additional features, we decided to build the system with a common front-end and back-end architecture. This separation circumvents the problem of cross-origin resource sharing (CORS) policies when accessing the file system directly through the front end, and further creates the possibility of running the back end on outsourced systems to match possible performance requirements.

The back end is based on Python and uses the Flask API framework to communicate with the front end through the IP network [Grinberg, 2018]. A new project is added to the platform's file system as a new folder, containing data sets, the model to inspect, and additional outsourced functions to allow custom data processing and visualizations. For the moment, Latent Inspector accepts file formats of .json, .npy, .mat, and .csv for the data sets and .h5 files for the network model. Every arising processing is done in the back end to minimize the front-ends performance requirement.

The front end uses Node.js along with Typescript as a basis and the PlotlyJS library for our main components to visualize data through interactive plots [Inc., 2015]. All together form a responsive and fluently scaling web application.

As shown in figure 3, Latent Explorer is divided into three parts following the idea of the above-mentioned functionalities: Starting with the setup process, the back end retrieves and forwards project information from the file system to the front end. After the user selection of the project and layer to inspect, the preparation process starts by splitting the model, propagating the dataset through the first sub-model, and reducing the dimensions with one of the provided techniques for visualization. Lastly, after the initial visualization is done, the user can interact with the model by applying inverse dimension reduction and propagating the user input through the remaining model layers.

All parts in the system have to follow one another, however, it is possible to jump back or iterate through parts of the system again if the user triggers another click event or modifies the current selections. Some minor features of Latent Inspector that do not affect the basic workflow are removed in figure 3 to resolve complexity.

The whole project is designed to be bundled into executable files. For the front end, we are using Electron as a wrapper around the web application to generate a cross-platform desktop application, and for the back end, we are using the python-based pyinstaller library. All relevant libraries to run the system are packed inside the executable to enable running on platforms without an integrated development environment.

4 Limitations and Outlook

4.1 Limitations

For the moment, Latent Inspector supports models with regression, classification, or combinational outputs. Since the tool needs to split the model at the inspected layer, it is only applicable for models with a causal, linear relationship of sequential layers. Models with branched connections like for instance the U-Net architecture with multiple direct input-to-output layer connections are rejected by the tool. However, models with localized branched connections such as ResNet may still be used since Latent Inspector can unite it as one layer without accessing the depth of the branches.

From the technical side, we require the model to be defined through the Keras Framework and saved in a .h5 file with all its trained weights [Chollet and others, 2015]. Furthermore, the loss of information when applying dimensional reduction techniques should be considered at any time. For PCA, the accuracy is shown in the UI and can drop rapidly if strongly high dimensional data sets are utilized. Speaking about the dataset, there is a noticeable loss in the Latent Inspector's front-end performance when aiming to plot more than 10,000 data points due to the high plotting precision that is currently applied.

4.2 Outlook

In the future, we will continue our generalized and versatile approach to support more machine learning architectures and development frameworks. Together with our modular system architecture, we plan to extend our tool with additional features like for instance interactive loss manipulation by dragging data points in the latent space [Wei *et al.*, 2022].

References

[Abadi *et al.*, 2015] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, 2015. Software available from tensorflow.org.

[Bank *et al.*, 2020] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[Chollet and others, 2015] François Chollet et al. Keras. https://keras.io, 2015. Accessed: 2023-02-07.

[Deng, 2012] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[Grinberg, 2018] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[Inc., 2015] Plotly Technologies Inc. Collaborative data science. https://plot.ly, 2015. Accessed: 2023-02-07.

[Jung and Oh, 2021] Hyungsik Jung and Youngrock Oh. Towards better explanations of class activation mapping. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1336–1344, 2021.

[Mohseni *et al.*, 2021] Sina Mohseni, Niloofar Zarei, and Eric D Ragan. A multidisciplinary survey and framework for design and evaluation of explainable ai systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 11(3-4):1–45, 2021.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Reiss and Stricker, 2012] Attila Reiss and Didier Stricker. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers*, pages 108–109. IEEE, 2012.

[Samek *et al.*, 2017] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[Wei *et al.*, 2022] Jiafu Wei, Haoran Xie, Chia-Ming Chang, and Xi Yang. Fine-tuning deep neural networks by interactively refining the 2d latent space of ambiguous images. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5948–5951. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Demo Track.

[Zhou *et al.*, 2023] Bo Zhou, Daniel Geißler, Marc Faulhaber, Clara Elisabeth Gleiß, Esther Friederike Zahn, et al. Mocapose: Motion capturing with textile-integrated capacitive sensors in loose-fitting smart garments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 7(1), March 2023.

[Ziegler, 1996] James F Ziegler. Terrestrial cosmic rays. *IBM journal of research and development*, 40(1):19–39, 1996.