Università Commerciale "Luigi Bocconi"
PhD School

PhD program in: Statistics

Cycle: XXXIII

Disciplinary Field (code): SECS-S/01

Interpretability & Explainability of Black-Box Models

Advisor: Emanuele BORGONOVO

PhD Thesis by

Roman Hahn

ID number: 3052555

Academic Year: 2022

Abstract

There is increasing need for methods that aid interpretability and explainability in Machine Learning (ML). In this thesis, inspired by recent studies calling for an increase in usage of Sensitivity Analysis (SA) methods, we explore interfaces between SA and ML. In particular, in our first work we use the notion of mean dimension to characterize the internal structure of complex modern neural network architectures. We exploit this notion with an innovative estimation routine which allows to calculate the mean dimension from data and we compare different architectures, comprising LeNet, ResNet and DenseNet.

In a second work, we analyze the difference between two trained neural networks using the notion of mean dimension and a new method for gaining interpretability on a dataset level, the so-called Xi-method. The networks arise from two different optimization routines, namely the standard stochastic gradient descent (SGD) and the recent replicated stochastic gradient descent (rSGD). The rSGD algorithm is designed to find flat minima in the loss landscape. The results, exploiting explainability tools, confirm findings of previous studies that suggest that the rSGD method is connected to better generalization.

Finally, we go on and ask the question if neural networks 'see' statistical dependence, and we offer a comparison between specific methods devised to understand the decision of neural networks and global sensitivity measures.

Overall, the thesis finds useful tools for investigation and shading light into modern black-box models.

Contents

Acknowledgements

In the course of my work, I have benefitted of help, advice, and support of many people. First of all, I would like to thank my Ph.D. advisor Emanuele Borgonovo for introducing me to the field of Sensitivity Analysis and for guiding my transition from student to researcher. He approved my work and consistently supported my efforts to gain confidence and overcome obstacles in my research. I am extremely grateful for his crucial role in my personal and scientific development. I thank Christoph Feinauer for sharing his knowledge in Machine Learning and Neural Networks. Working with him on the projects of this thesis has enriched the understanding in various ways and he has always been able to help me with any technical and programming-related problem that crossed my research path.

My thanks also go to the other collaborators and to the Bocconi Faculty that introduced me to fascinating topics in Statistics and provided solid foundations to my current and future research. My Ph.D. colleagues and especially my cohort freely shared new ideas, offered scientific and personal advice and were always encouraging especially during the harder times of the Ph.D.: I thank them all. Finally, I would like to thank my loved ones for helping me find my way and for their constant encouragement.

Chapter 1

Introduction

Machine learning tools, such as deep neural networks, have become the foremost method for a wide range of tasks. In applications such as image classification, speech recognition and natural language processing, neural networks show astonishing performance, i.e. predictive accuracy.

In 2016, Google *Deep Mind's Alpha Go* beat a top 10 ranked player in Go. The 26th move was the key to the later victory, though during the match it was not clear why this move could be advantageous. An expert stated

"It's not a human move. I've never seen a human play this move."

(Fan Hui, 2016).

This increase in performance is due to improvement of models and architectures, more and better data and higher computational power, mainly by the use of GPU's and parallel computing. But this increase in performance is not for free, usually these high-performing models come with a lack of interpretability and transparency.

Breiman (2001) emphasizes the trade-off between performance and interpretability. Since then it is a common belief that simpler models are easier to interpret but on cost of lower performance. This thesis will work towards weakening this statement by presenting and applying methods that bring light into the black-box algorithms. The goal of this

thesis is to present methods that maintain high performance but simultaneously provide explainability. I start with highlighting the need for interpretability. In many tasks such as playing *Go*, the key point is performance, while interpretability is a minor goal. However, there are many fields and applications where providing interpretability and explainability is inevitable (Samek et al., 2017). I present four key reasons highlighting the need for explainability.

Knowledge discovering

As these algorithms scan through masses of data, they might detect patterns unknown to humans. This learning from the system might be of special interest for example for physicists and chemists. They might get insight into hidden laws of nature. Or as earlier mentioned, *Go* players have now incorporated the move done by *Alpha Go* in their repertoire.

Model improving

Interpreting a model may also unveil its weaknesses, e.g. where the model fails, or where a decision suggested by the model is correct but said decision is based on wrong arguments or facts. Furthermore, interpreting might reveal some biases in the system. Knowing these weaknesses is the first step in trying to improve the model. This is especially important in preparation against adversarial attacks, where someone tries to fool the algorithm on purpose, for example in order to bypass spam-filters.

Verification & Trust

Interpretation goes hand in hand with trust. A decision maker cannot blindly trust predictions when she has no idea how the results came about. This is especially true in delicate fields such as medicine, where it is crucial to know why a certain diagnosis was made by the machine. A physician needs to verify the black-box output, such that she

understands how and why the output came about, and can follow the suggestion by the machine.

Legal Issues

With the omnipresence of artificial intelligence (AI) algorithms comes the demand for regulation. Decision makers are often obliged to be able to justify decisions. This plays a special role in fields of sensible information such as bank lending. For example, algorithms might learn some form of racism, hence persons with a specific name might have a much lower probability of getting a loan. The European Union adopted the 'right to explanation', a new regulation that allows users to ask for the reasoning of decisions made about her or his request.

Furthermore, and also related to trust, we still lack a proper uncertainty quantification. Take again the example of the field of medicine as pointed out by Begoli et al. (2019). The authors stress that these modern data rich black-box guise need to develop a principled and formal uncertainty quantification (UQ) discipline and refer to the success/development of uncertainty quantification in fields such as nuclear stockpile stewardship and risk management.

Fig. 1.1 illustrates the *data science rationale*. The first relationship represents the true one; found in *nature*. This is the relationship of interest that we want to make inference on or gain a better understanding of the behavior. The second layer represents the data level. In order to understand the nature and learn a model, one needs to collect data. Finally, the last part shows the machine learning algorithm. This algorithm is optimized or learned on the collected data. The goal is to train a model that resembles the behavior of nature as close as possible. But if the analyst's goal goes beyond prediction and she hasn't applied a white-box model, one that is interpretable by construction, she needs to

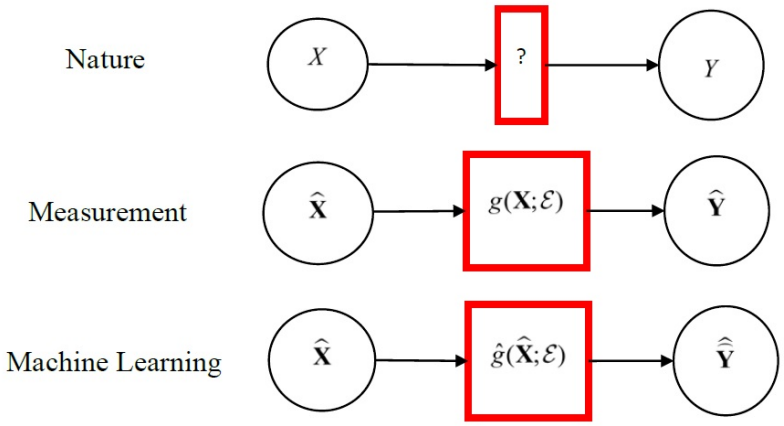


Figure 1.1: Data Science Rationale: From a conceptual model (layer 1), to the data level (layer 2) to the machine learning algorithm (layer 3).

fall back on methods that explain the model's decision.

In this PhD thesis, I explore, evaluate and develop such methods to explain and discover what is happening inside black-box models.

As we mainly focus on the model class of neural networks in the research works that form this thesis, I provide a concise introduction to this model class focusing on topics relevant for this thesis. Goodfellow et al. (2016) provides a thoroughly introduction to neural networks and deep learning.

Neural Networks are amongst the most common machine learning models. They are used for a great variety of tasks, such as classification, regression, or unsupervised learning. In particular, we will focus on a specific family of neural networks, namely deep learning models.

The famous *Universal Approximation Theorem* by Cybenko states that for any continuous function f on a compact set K (usually $\mathbb{R}^n$), there exists a feedforward neural network, having just one hidden layer, which uniformly approximates f within an arbitrary precision $\epsilon > 0$ on K. Hence, neural networks were traditionally built with only a single layer, arguing that if this layer is wide enough it can provide an accurate model.

However, a one layer model is characterized by a high bias, and sufficient training data

is required which capture all of the variations of the function. This fact and other beneficial properties for training a neural network made researchers starting to stack hidden layers. Stacking many of these hidden layers is then referred to as deep learning.

The data flow is typically from the input to the output layer, passing through the hidden ones - this procedure is known as forward propagation. There is a weight matrix W_l and a bias b_l (both combined in a parameter matrix w_l) associated to each layer l, as well as an activation function σ_l.

The parameters of a neural network are learned through the *backpropagation* algorithm, which is based on *Gradient Descent Learning* and is one of the main tools in Machine Learning optimization.

Learning a model requires to choose some hyperparameters, such as the architecture of the network, including how many layers the network should contain, how these layers should be connected to each other, how many units should be in each layer, and which activation functions should be employed. The backpropagation method requires the setting of some additional hyperparameters such as the learning rate and the optimizer for example.

The most common choices for activation functions are the sigmoid function, the hyperbolic tangent (TanH) and the Rectified Linear Unit (ReLU):

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \qquad \in [0,1] \tag{1.1}$$

$$\text{TanH}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad \in [-1,1] \tag{1.2}$$

$$\text{ReLU}(x) = \max(0,x) \qquad \in [0,\infty] \tag{1.3}$$

Different types of layer form a deep neural network. The following provides a concise description of popular layer types.

Fully Connected Layer These are identical to classical layers in standard neural networks. Each unit in a fully connected layer is densely connected to all the units of the previous one. The transformations applied to the data are a matrix multiplication (plus eventually a bias term) followed by the application of an element-wise nonlinear activation function σ, which results in $X^l = \sigma_l(W_l X^{l-1} + b_l)$. Since such a network is prone to overfit, several regularizers can be applied to its parameters including penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.).

Convolutional Layer These layers are designed to extract features from input data. They are often applied as first layers. In an image classification task, they preserve the relationship between pixels by learning image features using small squares of input images. They rely on the heuristic fact that the initial layers of a network detect low level features (for example, just vertical and horizontal edges) which are given as input to the next ones, while the deeper layers are able to capture more complex structures (like parts of the objects). Based on this, neurons in convolutional layers are not fully connected to every single pixel in their input (like they would be in standard neural networks), but only to localized areas, made of pixels with a limited spatial extent called Local Receptive Fields. For this aim, so-called convolution kernels (or filters) slide along input features and provide responses known as feature maps.

Mathematically the discrete convolution operation of input I using filter K of size $2h_1 + 1 \times 2h_2 + 1$ can be written as

$$(K * I)_{i,j} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{i-u,j-v},$$

where the filter K has the following form

$$K = \begin{Bmatrix} K_{-h_1,-h_2} & \dots & K_{-h_1,h_2} \\ \dots & K_{0,0} & \dots \\ K_{h_1,-h_2} & \dots & K_{h_1,h_2} \end{Bmatrix}.$$

For each layer l, one chooses how many filters to apply. The chosen number m_1^l is equivalent to the depth of the volume of output feature maps. The number of input channels and output channels are hyper-parameters. Moreover, convolutional filters/kernels are defined by a certain width and height. The idea is that each filter should detect a particular feature at every location on the input. The output X_i^l of layer l consists of m_1^l feature maps of size $m_2^l \times m_3^l$. The i^{th} feature map, denoted X_i^l, is computed as

$$X_i^l = B_i^l + \sum_{j=1}^{m_1^{l-1}} K_{i,j}^l X_j^{l-1},$$

where B_i^l is a bias matrix and $K_{i,j}^l$ is the filter connecting the j^{th} feature map in layer $l-1$ with the i^{th} feature map in layer l.

See Goodfellow et al. (2016) for a description of the additional hyperparameters of the convolution operation such as padding, stride, and dilation. They are omitted, since they are not needed for the conceptual understanding of the convolution operation.

Three main ideas lead to the use of convolutions: *sparse interactions*, *parameter sharing* and *equivariance to translation*. I provide a quick description here. Detecting low level features such as edges does not require the possible millions of interactions of pixels but only pixel interactions of locally connected ones; this **sparse connectivity** is represented using the convolution with a kernel smaller than the input. This leads to the storage of fewer parameters, reducing memory requirements and statistical efficiency of the model (preventing the model from overfitting).

Parameter sharing is the second advantage, which in turn leads directly to *equivariance*

of translation. In a traditional neural network a single weight is used exactly once, in CNNs each member of the (same) kernel is used at every position of the input, removing the needs of estimating one set of parameters for every location. A function $f(x)$ is said to be equivariant to g if $f(g(x)) = g(f(x))$. Hence in an image classification problem, this translates to the detection of an object no matter its position.

Pooling Layer A pooling layer operates on blocks of the input feature map and combines the feature activations locally by means of a pooling function. Common choices are the average or the max function. As the names suggest, with max pooling operation, the maximum activation is taken from the selected block of values, while with mean pooling the choice is the mean of the activations. The size of the pooled region and the stride must be specified as hyperparameters, similarly to the convolution layer. The window is then slided across the input feature maps with a step size defined by the stride. Pooling layers reduce the dimensions of data by effectively downsampling the input feature map. The obtained compact feature representation is invariant to moderate changes in object scale, pose, and translation in an image for example.

In this thesis, we mainly focus on convolutional neural networks (CNNs). In a typical CNN, convolutional layers are usually placed in the beginning and fully connected layers are placed at the end of the architecture. CNNs take a different approach towards regularization compared to standard neural network architectures: they take advantage of the hierarchical pattern in the data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, CNNs have much less connections as fully connected layers and can be seen as regularized versions of multilayer perceptrons.

The underlying and common frame of my thesis projects is to develop and extend methods based on global sensitivity analysis (GSA) for the purpose of explaining decisions by complex machine learning models.

These methods are usually post-hoc explanations as defined by Murdoch et al. (2019), as such explanation methods are applied to the already trained machine learning models. Thus, post-hoc explanation methods aim to explain models decision where the inner structure is already fixed.

There are arguments against such post-hoc explanations. For example, Rudin (2019) argues to use interpretable models instead of trying to explain the decision of black-box models as post-hoc explanations are often not reliable, and can be misleading.

While an interpretable model is indeed the better choice for many applications, there are applications where interpretable models reach a level of accuracy far from the one of black-box models, such as deep neural networks. One such application is image classification. Since the recent advances in computational power and architecture designs, convolutional neural networks are the most common and most successful method for classifying images. In this thesis, I focus mainly on image classification examples.

I intend to demonstrate the usefulness of sensitivity analysis (SA) employed and embedded in methods to increase interpretability of black-box models. A concise overview of the related literature in this framework is given here. In the most general sense, SA is the study of how the outputs of a system are related to, and are influenced by, its inputs. While SA is becoming more and more an integral part of mathematical modeling, the following paragraph focuses on SA in the context of machine learning. There is a wide research field that applies SA to feature and structure selection. Though this is indeed a very interesting research stream, see Razavi et al. (2021) for an overview, we concentrate here to its application for its use in interpretability and explainability of ML-models. I do not claim completeness here and only describe few works in this context, mainly referring to literature that is not mentioned in the chapters of this thesis but have contributed noticeable to the field.

The robustness of decision boundaries for classification with respect to data and/or model hypotheses can be explained and examined using SA. A sensitivity index developed

by Lemaître et al. (2015) is applied by Bachoc et al. (2020) for robustness analysis of decision boundaries in classification. Moreover, Spagnol et al. (2019) and Molnar (2019) identify influential inputs regarding the occurrence of critical events, again related to the robustness analysis of decision boundaries. Owen (2014) and Lundberg and Lee (2017) have independently developed importance measures being able to deal with dependent inputs/features using Shapley values (Shapley, 1953). In Lundberg et al. (2019), an efficient method for estimating Shapley values for regression trees is developed.

Model explanation methods can broadly be categorized into three categories: Explanations on the individual input level, explanations on the dataset level, and methods for trend identification. Representatives of the first category are *Layerwise Relevance Propagation* (LRP) (Bach et al., 2015) and *Local Interpretable Model-agnostic Explanations* (LIME) (Ribeiro et al., 2016). LRP is an explanation method designed for neural networks and is described in greater detail in the last chapter, while LIME is model-agnostic and so compatible with many different classifiers. LIME manipulates the input data and creates a series of artificial data containing only a part of the original attributes. The new data is then evaluated by the model and weighted by their proximity to the original example. Then, an interpretable model is learned on the perturbed data points and the associated prediction. It can be used with text, image and tabular data. To the second group belong works such as *Feature Permutation Importance* of (Breiman, 2001) or the Shapley-value based method of Lundberg and Lee (2017). We provide a new method to this category in Chapter 3. Noteable studies on the third group of literature include *partial dependence function* of Friedman (2001), ICE plots (Goldstein et al., 2015), and ALE plots (Apley and Zhu, 2020) form the category of trend identification.

A popular tool to get insight in the behavior of a black-box model is the partial dependence plot. For the purpose of illustration, this paragraph presents an empirical analysis of the use of partial dependence plots and the so-called "one-way functions" from Borgonovo et al. (working paper 2019+) in order to gain insights in black-box

models. Originally presented in Friedman's 2001 paper *Greedy Function Approximation: A Gradient Boosting Machine*, partial dependence plots are applicable to all kind of supervised prediction models. Let $(\mathbf{x}^i, y^i)$ with $i = 1, ..., N$ be the training data and $g : \mathcal{X} \to \mathbb{R}$ the learned predictive model, mapping from input space $\mathcal{X}$ to $\mathbb{R}$. Formally, a partial dependence function is defined as

$$h_i(x_i) = \int_{\mathcal{X}_{-i}} g(x_i, x_{-i}) dF_{x_{-i}}(x_{-i}), \tag{1.4}$$

where $(x_i; x_{-i})$ is a point in $X = X_i \times X_i$ and x_{-i} denotes all variables but x_i. The probability distribution of X is denoted by F_X.

Borgonovo et al. (working paper 2019+) study the properties of partial dependence functions formally, to understand whether partial dependence functions preserve properties such as monotonicity and convexity present in the original input-output mapping. Moreover, they disaggregate partial dependence plots by introducing the so-called "one-way functions" that work in a similar way as the "ICE curves" by Goldstein et al. (2015). Both show why these additional curves help understand the model behavior when inputs are dependent. Considering the average partial relationship between the input variable and the output (averaging over all other inputs) can misguide the decision maker to wrong conclusions. When considerable interaction effects are present, the partial dependence plots might not capture heterogeneous relationships due to averaging over all inputs. Formally, a one-way sensitivity function is a curve of the type

$$w_i^0(x_i) = g(x_i, x_{-i}^0); \qquad w_i^0 : \mathcal{X}_i \to \mathbb{R}. \tag{1.5}$$

Thus, a one-way sensitivity function, instead, examines the impact on the output while changing the value of the specified input and holding all other inputs constant at a specific/observed configuration $\mathbf{x}_{\sim i}^0$.

My PhD thesis is structured into four chapters beginning with this introduction. The

other chapters are stand alone research projects. They are not necessarily related but have all the common aim of shading light into black-box models. The chapters present research works that are or will be submitted to peer-reviewed journals.

Chapter 2 presents the work **The Mean Dimension of Neural Networks and what it reveals** which is joint project with my supervisor Emanuele Borgonovo and Christoph Feinauer. The generalization performance of artificial neural networks is increasing their success in several complex tasks. However, the lack of a theoretical explanation of their inner mechanisms forces us to regard them as black boxes. A crucial information about their inner structure is the size of interactions a neural network detects in the data. Recent works argue that the notion of mean dimension provides us with a relevant tool to open up a neural network black box and facilitate cross comparisons among network architectures. We show that we can estimate the mean dimension from a given dataset, without the need of resampling from a hypothesized distribution. We then use the mean dimension to follow the evolution of interactions during training, to analyse how interactions propagate layer by layer in a network and to understand how the type of activation function impacts the magnitude of interactions once the network structure is fixed. We carry out experiments on synthetic datasets as well as experiments on the CIFAR-10 image dataset, analysing LeNet and more modern and complex architectures like ResNet and DenseNet.

Another joint project with my supervisor Emanuele Borgonovo and Christoph Feinauer is the work with the title **Using Tools for Interpretability for a Comparison of Neural Network Optimization Routines** presented in the Chapter 3. The high dimensional parameter space of deep neural networks makes its optimization a non-trivial task. Recent research suggests that flat minima in the empirical risk landscape of neural networks possess better generalization capabilities with respect to sharp ones. We are comparing neural networks trained using a standard stochastic gradient and the so-called replicated stochastic gradient method from a sensitivity analysis angle. The latter algorithm is particularly developed to find flat minima. The experiments are based on different

well-known architectures (LeNet, ResNet and DenseNet) applied to an image classification task using the CIFAR-10 database. We introduce a modified version of the Xi-method for black-box model explanations on a dataset level and exploit the recent methodology of mean dimension of neural networks for further insights in the differences between the models trained with the two optimization algorithms. Our results show that although the accuracy of the used models is quite similar, there are significant differences in the model sensitivities indicating indeed a better generalization of the models trained with the replicated stochastic gradient optimization algorithm.

Chapter 4 asks the question: **Neural Networks and Statistical Dependence: Does it Matter?**. This is a collaboration with Emanuele Borgonovo, Elmar Plischke, Christoph Feinauer and Valentina Ghidini. Deep neural networks have become the most popular method for many complex prediction tasks such as image classification and natural language processing. Although the functional form of a single neuron is quite simple, the huge number of neurons and graph-like structure of neural networks make them become black-box models. Hence the inner mechanics of such models are unknown to its user. In particular, which pixels drive a neural network's decision in an image classification task is concealed. We ask if features that are statistically important for the output are also relevant for the neural network's decision. We introduce probabilistic sensitivity measures designed for classification tasks that give importance to pixels solely on the input-output mapping. An aggregated version of the well-known Layerwise Relevance Propagation provides importance scores for the pixel based on their importance for the network's decision. We introduce further methods of 'feature importance' for comparison. The findings are evaluated using degradation plots. The results suggest that what is statistically relevant for the input-output mapping is also relevant for the neural network classification.

References

Apley, D. W. and Zhu, J. (2020). Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, forthcomin.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K. R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7):1–44.

Bachoc, F., Gamboa, F., Halford, M., Loubes, J.-M., and Risser, L. (2020). Explaining machine learning models using entropic variable projection.

Begoli, E., Bhattacharya, T., and Kusnezov, D. (2019). The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence*, 1(1):20–23.

Breiman, L. (2001). Statistical modeling: The two cultures. *Statistical Science*, 16(3):199–215.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Lemaître, P., Sergienko, E., Arnaud, A., Bousquet, N., Gamboa, F., and Iooss, B. (2015).

Density modification-based reliability sensitivity analysis. *Journal of Statistical Computation and Simulation*, 85(6).

Lundberg, S., Erion, G., and Lee, S.-I. (2019). Consistent Individualized Feature Attribution for Tree Ensembles. *ArXiv*, arXiv:1802:1–9.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 4766–4775.

Molnar, C. (2019). Interpretable Machine Learning. A Guide for Making Black Box Models Explainable. *Book*.

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. (2019). Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences of the United States of America*, 116(44).

Nuclear Energy Agency (1989). PSACOIN Level E Intercomparison. Technical report.

Owen, A. B. (2014). Sobol' Indices and Shapley Value. *SIAM/ASA Journal on Uncertainty Quantification*, 2(1):245–251.

Razavi, S., Jakeman, A., Saltelli, A., Prieur, C., Iooss, B., Borgonovo, E., Plischke, E., Lo Piano, S., Iwanaga, T., Becker, W., Tarantola, S., Guillaume, J. H., Jakeman, J., Gupta, H., Melillo, N., Rabitti, G., Chabridon, V., Duan, Q., Sun, X., Smith, S., Sheikholeslami, R., Hosseini, N., Asadzadeh, M., Puy, A., Kucherenko, S., and Maier, H. R. (2021). The Future of Sensitivity Analysis: An essential discipline for systems modeling and policy support. *Environmental Modelling and Software*, 137(December 2020).

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, volume 13-17-Augu, pages 1135–1144.

Rudin, C. (2019). Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 1(5):206–215.

Saltelli, A. (2002). Sensitivity Analysis for Importance Assessment. *Risk Analysis*, 22(3):579–590.

Samek, W., Wiegand, T., and Müller, K.-R. (2017). Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. *ArXiv*, arXiv:1708.08296.

Shapley, L. S. (1953). A Value for n person Games. Contributions to the Theory of Games. *Annals of Mathematics Studies*, 28(2).

Spagnol, A., Le Riche, R., and Da Veiga, S. (2019). Global sensitivity analysis for optimization with variable selection. *SIAM-ASA Journal on Uncertainty Quantification*, 7(2).

Chapter 2

The Mean Dimension of Neural Networks and What it Reveals

2.1 Introduction

While successes of machine learning models based on deep neural networks have revolutionized fields like computer vision (Voulodimos et al., 2018), natural language processing (Young et al., 2018) and reinforcement learning (Li, 2017), there is currently no theory that can explain their empirically observed generalization performance or give a satisfactory answer to the question on how it is achieved (Sejnowski, 2020).

Exacerbating the problem is the growing diversity of architectures, optimization techniques and data preprocessing pipelines: While the variety of different design choices that neural networks allow for gives researchers many possibilities for tinkering and should be considered an advantage if the only goal is predictive performance, it is often not clear what contribution of the single parts to the final model is. In addition, the large computational resources needed for training and inference of state-of-the-art networks often prohibit ablation studies, hinder comparison of models and render the application of standard machine learning tools like cross-validation difficult.

In classification, the central result presented is typically the accuracy of predictions. While this metric allows for competitive comparison between models, it says nearly nothing about how other properties, like differential sensitivity to inputs or the richness of internal representations, might compare between the models. In this work, we argue that other metrics for assessing the properties of different models, and also of the same model during the training trajectory, are needed.

Connected to the difficulty of comparing neural network models is their infamous *black-box* nature: While the single neuron is mathematically simple, the input-output relationship that results from the combination of millions of such units is a highly complex function. This, in turn, makes the *interpretation* and *explanation* of neural network decisions hard (Montavon et al., 2018), a problem that is not helped by the observation that there is no consensus on what qualifies as an explanation (Lipton, 2018).

This fundamental lack of transparency and comparative metrics has practical implications: First, additional ways of analysing and comparing models might lead to a more principled approach to improving them. Secondly, the adaption of neural networks might be delayed due to a hesitation to accept advice from a model that is fundamentally unexplainable (London, 2019), and for which even an analysis of possible failure modes is lacking.

In this work, we present two novelties that were, to the best of our knowledge, not introduced so far: We exploit the concept of the effective mean dimension of interaction (Owen, 2003a) as a metric for comparing different architectures of neural networks and also apply it to analyze in which layer interactions occur. The mean dimension metric is rooted in the domain of global sensitivity analysis and describes the average size of interaction terms in a functional decomposition of the input-output relationship of the neural network. The second novelty is the inverse PCA neural network layer, which can be used with any method that requires decorrelated inputs. This layer can be added *after* the neural network has been trained, and does not change the predictions.

Recently, Hoyt and Owen (2020b) investigate routines for computing the mean dimension of neural networks. We demonstrate how the notion sheds light into the black-boxes of common neural network architectures. We are interested in using the mean dimension to answer the following questions: what is the expected size of interactions that a neural network looks at? Where do interactions arise in neural networks? How does the interaction size evolves during the layers? What are the effects of different activation functions on the size of interactions in a neural network?

The work is structured as follows: In Section 2.2, we review the related literature. In Section 2.3.1, we present the notion of mean dimension and some related concepts. In Section 2.3.2, we are presenting our adaption of the methodology for neural networks. In Section 2.4 and 2.5, we present our numerical results, starting with a layer by layer analysis of the development of mean dimension and continuing with an analysis of the evolution during training in an image classification task. In Section 2.6, we discuss them and point for further research directions.

2.2 Related Literature

In recent years, neural networks have become the state of the art in many machine learning applications. Deep neural networks show astonishing performance on complex tasks such as speech recognition, visual object recognition and many other domains such as computational biology. We refer to Lecun et al. (2015) for a review. However, the complex architectures needed to solve such problems make the interpretation of single decisions/predictions of neural networks difficult and they need to be regarded as *black-box* models. Interpretability of neural networks is a topical subject of investigation. The review of Guidotti et al. (2018) analyzes various needs and forms for interpretability in complex machine learning tasks. Others, such as Begoli et al. (2019), emphasize the necessity of uncertainty quantification in deep learning and the lack of theory in the new data-driven

methods. They also argue for the development of a principled and formal uncertainty quantification discipline for deep learning, for example in medical applications. This is echoed by Rudin (2019) who favors for this reason simple and interpretable models when possible. Breiman (2001) is among the first to emphasize the trade-of between performance and interpretability.

Numerous approaches have appeared in the literature to provide explanations in neural network decisions. Methods for interpretability can be broadly divided into three categories: The first category is on the individual prediction level. Methods such as "LIME" proposed in Ribeiro et al. (2016) or such as "Layerwise Relevance Propagation" in Bach et al. (2015) belong to this category. The second category is at the dataset level. To this group belong methods based on permutation, such as Feature Permutation Importance of (Breiman, 2001) or the Shapley-value based method of Lundberg and Lee (2017). The third category comprises trend identification methods, such as partial dependence function of Friedman (2002), ICE plots (Goldstein et al., 2015), and ALE plots (Apley and Zhu, 2020).

Closely connected to the explainability of input-output mappings is the analysis of interactions. Broadly speaking, this refers to the idea that the output can be explained by assigning the input variables to interacting groups of different sizes and calculating the output based on these groups. In statistics such intuition originates back in the 1920s, when Fisher introduced the two-way ANOVA. One can perform either individual tests for interaction such as in Fishers two-way table and additive groves or one explicitly specifies the interaction of interest and runs a lasso approach (Tibshirani, 1996) to find the important interactions. The specific problem of the detecting pairwise interactions is closely connected to the problem of inferring the edge topology of graphs and has important applications in physics (Nguyen et al., 2017) and biology (Cocco et al., 2018).

It is natural to ask whether interaction analysis methods can help us obtaining insights on the structure of deep learning models. However, most standard methods for the calculation of interactions are out of reach due to the exponential number of interaction

terms. Recent research tackles this problem for example by interpreting the learned weights of a feed-forward neural network in order to detect interactions (Tsang et al., 2018). Tsang et al. (2018) find that interactions between input features stem from the non-additive effect generated by the presence of nonlinear activation functions. The main question posed in these works is which features are part of which interactions.

At the same time, methods to understand the structure of a multivariate input-output mapping have been developed in the machine learning literature using the functional Anova expansion (see Rabitz and Aliş (1999) for a review). Using the functional Anova decomposition, one can expand a multivariate mapping into 2^N terms, in which second order and higher order terms account for the residual level of interaction. Building on that representation, the classical results of Efron and Stein (1981) and Sobol' (1993) allow us to decompose the prediction variance as a sum of variance contribution by all subgroups of indices. In Caflisch et al. (1997) and Owen (2003b), the functional Anova expansion is exploited for introducing the notion of dimension distribution of a function, which in turn leads to the notion of mean dimension. The mean dimension provides an indication about the average magnitude of interactions in a neural network, and can help to shed light on its inner structure and facilitate cross comparisons.

The interest in computing the mean dimension is reflected in the recent work of (Hoyt and Owen, 2020b) that addresses several computational aspects. However, the notion has not yet been fully explored. The goal of this paper is to fill this gap, providing a new way of calculating the mean dimension and presenting several experiments that highlight a summary of insights that can be obtained from the mean dimension of a neural network.

2.3 Methods

2.3.1 The Mean Dimension

The notion of mean dimension is introduced in Caflisch et al. (1997) and based on a central result in statistics, the functional ANOVA expansion. Let $(\Omega, \mathcal{B}(\Omega), \mathbb{P})$ be a reference probability space. We consider features as a multi-variate random vector $X = \{X_1, X_2, \ldots, X_d\}$ on $(\Omega, \mathcal{B}(\Omega), \mathbb{P})$ with cumulative distribution function $F_X(x)$. We also consider the target Y a random variable (or vector) as produced by a predictive model, $g : R^d \to R$, with $Y = g(X)$.

Assuming that g is square integrable, and that $F_X(x)$ is a product measure, Efron and Stein (1981) prove the classical functional ANOVA representation (Sobol', 1993; Rabitz and Aliş, 1999; Owen, 2003b):

$$g(x) = \sum_{u \subseteq \{1,2,\ldots,d\}} g_u(x_u), \tag{2.1}$$

where the component functions $g_u(x_u)$ are called functional ANOVA effects and are computed from

$$g_u(x_u) = \int_{X_{\sim u}} g(x) dF_{\sim u}(x_{\sim u}) - \sum_{v \subset u} g_v(x_v), \tag{2.2}$$

where the symbol $\sim u$ represents the set of all indices with exclusion of the ones in u. We set $g_\varnothing = E[g(X)]$. Correspondingly, F_u is the cumulative distribution function of the inputs with indices in u and $F_{\sim u}$ is the cumulative distribution function of all inputs but the ones in u. It is possible to prove that the component functions are orthogonal, that is, given two subset of indices u and v, $\int g_u(x_u) g_v(x_v) dx = 0$ as long as $u \neq v$, and that their mean value is null, i.e., $\int g_u dx_u = 0$. One can then decompose the variance of $g(X)$, σ^2, in $2^d - 1$ terms, writing:

$$\sigma^2 = \sum_{u \subseteq \{1,2,\ldots,d\}} \sigma_u^2, \tag{2.3}$$

where

$$\sigma_u^2 = \int_{X_u} \left[g_u(\mathbf{x}_u)\right]^2 dF_u(\mathbf{x}_u). \tag{2.4}$$

The effects σ_u in Equation 2.4 are directly related to the variance of the corresponding effect functions, and capture the contribution of the residual interactions among the features whose indices are in u to the overall variance σ^2.

One defines an auxiliary probability mass function over all subsets u of indices whose values are given by

$$S_u = \frac{\sigma_u^2}{\sigma^2}. \tag{2.5}$$

Then, the mean dimension is defined as ((Caflisch et al., 1997), (Owen, 2003b))

$$D_g = \sum_{u \subseteq \{1,2,\ldots,d\}} |u| \frac{\sigma_u^2}{\sigma^2}, \tag{2.6}$$

where $|u|$ is the cardinality of u. Thus the value of D_g is the average of the cardinality of the subsets of all indices weighted by their fractional contribution to the variance. Note that the mean dimension is a single number providing an average information about the size of interactions: It does not deliver detailed information of what specific interactions are important.

There are two general results that we employ in this work to define our estimation procedure. The first is the relationship between the mean dimension and the total order variance-based sensitivity indices of Homma and Saltelli (1996), proven by Owen (2003b).

These indices are defined as

$$\overline{\tau}_i^2 = \sum_{v: i \in v} \sigma_v^2, . \tag{2.7}$$

The total index $\overline{\tau}_i^2$ represents the overall fraction of the variance of the target contributed by X_i. Owen (2003b) shows that the numerator of D_g equals the sum of $\overline{\tau}_i^2$ over all features that is $\sum_u |u| \sigma_u^2 = \sum_{i=1}^d \overline{\tau}_i^2$. The second is the following. Let $X^0 = \{X_1^0, X_2^0, \ldots, X_i^0, \ldots, X_d^0\}$ and $X^1 = \{X_1^0, X_2^0, \ldots, X_i^1, ..., X_d^0\}$ be two points in the input space that differ only in the

i^{th} feature dimension. Then, take the finite difference $\Phi_i(X_i^1) = g(X^1) - g(X^0)$. Consider then the evaluation of such finite difference at randomized locations in the feature space. It is possible to show (Borgonovo and Rabitti, 2020), that half the variance of this population is equal to the total index of X_i, that is

$$\overline{\tau}_i^2 = \frac{\mathbf{V}[\Phi_i(X_i^1)]}{2}, \tag{2.8}$$

where $\mathbf{V}[A]$ stands for the variance of A. Then, it follows that

$$D_g = \frac{\sum_{i=1}^{d} \mathbf{V}[\Phi_i(X_i^1)]}{2\sigma^2}. \tag{2.9}$$

The above equations can then be translated into estimators for the mean dimension from given data. We discuss these aspects in the next sections.

2.3.2 Estimating the Mean Dimension from a Given Dataset

Suppose we have a labelled dataset of inputs x and a neural network trained on this data. Let $x^k = (x_1^k, ..., x_d^k)$ denote the k^{th} input with $k = 1, 2, \ldots, N$, where N is the sample size. Let also $y^k = g(x^k)$ be the network prediction corresponding to input x^k. We then consider modified samples where a single feature value has been replaced with the same feature value from a different input. We denote such modified inputs as

$$(x_i^l; x_{\sim i}^k) = (x_1^k, \ldots, x_{i-1}^k, x_i^l, x_{i+1}^k, \ldots, x_d^k), \tag{2.10}$$

where the sequence $(x_i^l; x_{\sim i}^k)$ is created by replacing x_i^k with x_i^l in x^k. The finite change with respect to this modification is then defined as

$$\phi_i^{kl} = g(x_i^l; x_{\sim i}^k) - g(x^k). \tag{2.11}$$

These finite changes are computed for r pairs of inputs, x^k and x^l, uniformly sampled from the dataset. This yields an $r \times d$-matrix of finite differences, with r finite changes for each input feature.

We now consider that we have trained a neural network g for the input-output mapping. The previous equality results in the following estimator for the total index $\overline{\tau}_i^2$:

$$\hat{\overline{\tau}}_i^2 = \frac{\hat{\sigma}^2[\phi_i(X_i^1)]}{2} = \sum_{k=1}^{r} \frac{(\phi_i^{k,l} - \hat{\mu}_{\phi_i})^2}{2(r-1)}, \tag{2.12}$$

where $\hat{\sigma}^2[\phi_i(X_i^1)]$ is an estimator of $\mathbf{V}[\Phi_i(X_i^1)]$ and $\hat{\mu}_{\phi_i}$ is an estimator of the mean of the population of the finite differences Φ. In Borgonovo and Rabitti (2020), it is shown that this population has zero mean. Thus, one expects this quantity to be close to zero at sufficiently large sample sizes. Then, by Equation (2.9), we obtain the following estimator for the mean dimension:

$$\hat{D}_g = \sum_{i=1}^{d} \frac{\hat{\overline{\tau}}_i^2}{\hat{\sigma}^2} = \sum_{i=1}^{d} \left\{ \sum_{k=1}^{r} \frac{(\phi_i^{k,l} - \hat{\mu}_{\phi_i})^2}{2(r-1)\hat{\sigma}^2} \right\}, \tag{2.13}$$

where $\hat{\sigma}^2$ is an estimator of the output variance σ^2. That is, computing a set of $N-1$ finite differences of network predictions and taking their variance, we obtain an estimate of the total index of X_i. Then, summing these estimates and dividing by the estimated output variance, we obtain an estimate of the mean dimension of g.

2.4 Experiments with Synthetic Data

In this section, we propose a series of experiments on a test case of small dimensionality to illustrate the key concepts and some inferences that can be made with the mean dimension.

2.4.1 An Analytical Test Case: Ishigami

The first experiments are centered around a regression problem with a scalar output. We use a dataset generated from the well known Ishigami function (Ishigami and Homma, 1990):

$$g(x_1, x_2, x_3) = \sin(x_1) + 7\ \sin^2(x_2) + 0.1\ x_3^4 \sin(x_1).$$

The features x_j are uniformly and independently distributed between $-\pi$ and π for $j = 1, 2, 3$. This function is often used as a test for uncertainty and sensitivity analysis methods.

The mean dimension is known analytically for this model (Kucherenko et al., 2015) and is equal to $D_{\mathrm{f}} = 1.24$. We used a small feed-forward neural network with two layers for the following experiments. We refer to the appendix for details on the experimental settings.

The estimation routine from Section 2.3.2 yields a very precise estimate of the mean dimension for any of the used activation functions. Using a mean of 20 replicates, we obtain an average mean dimension of 1.241 with a standard deviation of 6.6×10^{-3} using the Rectified Linear Unit (ReLU) activation function. Thus, in terms of the mean dimension, the learned neural network is a very accurate meta-model of the Ishigami function.

Fig. 2.1 shows the estimated mean dimension and mean squared error at increasing sample sizes. The mean dimension approaches 1.24 already at a sample size of about $N = 200$. — N here refers, on the one hand, to the sample size used to train the model, and on the other hand to the number of pairs used to estimate the mean dimension (r in Section 2.3.2), as in our experiments we use $r = N - 1$. —

We stress these findings in two ways. First, we repeated the experiments using the hyperbolic tangent (TanH) activation function, obtaining very similar results. Then, we added noise to the data by augmenting the feature set with 7 dummy variables, for a total input dimension $d = 10$. The additional features are random replicates of the three true

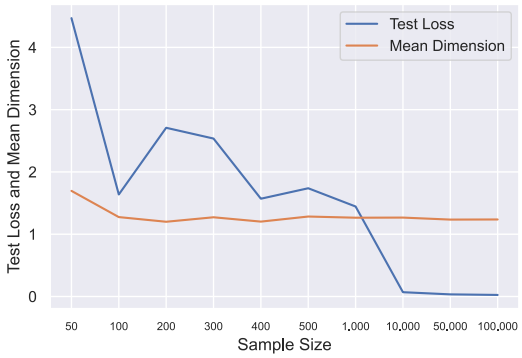
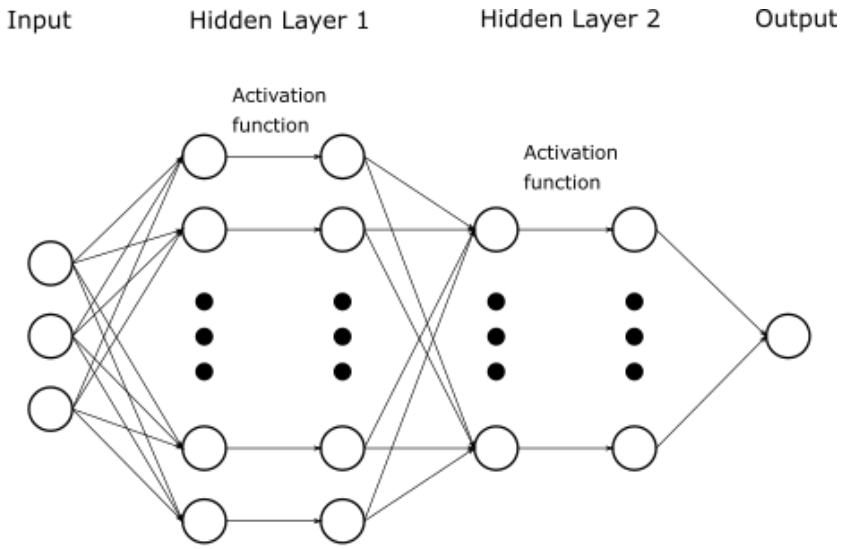


Figure 2.1: Mean dimension estimate (MD) and mean square error (loss) as the sample size increases for a small feed-forward neural network with ReLU activation function, trained on the Ishigami function. Note that the number of finite differences used for estimating the mean dimension scales linearly with the sample size.

inputs, drawn uniformly between $-\pi$ and π and independent of the output y. We keep the original output in the dataset. Retraining the network on these noisy data and estimating the mean dimension leads to results again similar to the ones we just illustrated.

2.4.2 Where Do Interactions Occur?

To study how the mean dimension changes throughout the network, we regard each node in the network as an output and calculate the mean dimension for each neuron. Fig. 2.2 provides a visualization of the results and a sketch of the architecture. Each neuron in Fig. 2.2 has a corresponding mean dimension. In the table below the network architecture, however, we report the layer average mean dimension (LAMD). Also, for each layer we consider the mean dimension before and after activation (to illustrate the numbers 1.008 and 1.1891 refers to the LAMD of the first layer without and with activation, respectively). We perform this layer-by-layer analysis for a neural network trained on the Ishigami data (Section 2.4.2.1) and for a random neural network using random data as input (Section 2.4.2.2).



Average mean dimension

Ishigami - ReLU	1.0008	1.1891	1.0403	1.2017	1.2413
Ishigami - TanH	1.0010	1.0687	1.1032	1.3606	1.2416
Random - ReLU	1.0002	1.4273	1.4255	2.1308	1.8615
Random - TanH	1.0001	1.0491	1.0499	1.0549	1.0528

Figure 2.2: Sketch of the neural network architecture used for the Ishigami experiments (Section 2.4.1 and 2.4.2.1) & the random model and random data experiment (2.4.2.2). Below: Table with average of 20 replicates of the per layer average of mean dimension for different experiment set ups and different activation function (ReLU and TanH). The layer architecture has the following structure: fully-connected layer, activation function, fully-connected layer, activation function, fully-connected layer.

2.4.2.1 Neural Network Trained on Ishigami Data

The table in Fig. 2.2 shows the average over 20 replicates of the LAMD. The first two rows show the LAMD values using the Ishigami data for a network with ReLU activation and with TanH activation, respectively. The even columns report results without activation, the odd columns after activation. One notes the strong increase in LAMD following an activation. For instance going from column 1 to column 2 in the first row (with ReLU activation), we observe a systematic increase in the LAMD of the first layer. The same holds for the second application of activation function, (column 3 to column 4). While this is consistent for both activation functions, the ReLU activation gives a higher increase in the LAMD of the first layer, while TanH does so in the LAMD of the second layer.

There, note that the mean dimension of the second layer on the Ishigami data with TanH activation is higher than the output mean dimension. Because the output mean dimension is the same with both activation functions, the LAMD has then to decrease in order to get to the correct estimate.

Overall, interactions seem to arise due to the non-linear activation functions, while fully-connected layers (pre-activation) do not lead to increases in the mean dimension. We also get a mean dimension very close to 1 for each pre-activation neuron in the first layer, because the first layer without activation is a linear transformation of the inputs.

2.4.2.2 Random Neural Network & Random Input

In the next experiments, we are interested in determining the mean dimension of a random neural network with random inputs and different activation functions. We use the default Kaiming initialization of Pytorch (He et al., 2015) for the weights and draw the inputs as i.i.d. samples from a standard normal distribution. We generate a sample of size 60.000 with an input dimension of 200. The neural network architecture differs from the previous section only in the size of the incoming connections from the input layer (see Fig. 2.2). Activation either with ReLU or TanH increases the mean dimension with respect to the non-activated case. The third row of the table in Fig. 2.2 shows that the mean dimension of g with ReLU activation functions is 1.86 when averaging over 20 replicates. The average mean dimension of the same experiment using TanH activation function is 1.05. This indicates that when using random data, the ReLU activation function induces higher average interaction size than the TanH activation function. These results are in line with the experiments performed in Section 2.4.2.1.

2.5 Image Classification

We now apply the analysis to an image classification problem using neural networks. We use the CIFAR-10 data set (Krizhevsky, 2009), which contains 60.000 labeled images with 10 different labels. The images consist of 32×32 pixels with 3 color channels. Two issues arise when trying to estimate the mean dimension following the routine in Section 2.3.2. First, the estimation of the mean dimension requires a scalar output, while a classification problem usually yields a set of probabilities, one for each class. Second, as remarked in Section 2.3.2, the estimation procedure requires independent features. The correlation matrix for real world images shows very high correlations, especially between neighboring pixels. Thus, we cannot estimate the mean dimension of an image classification task using the original inputs.

To address this second issue, we consider now PCA-transformed images as inputs, creating a set of uncorrelated features, and we extend the network architecture by inserting an inverse-PCA transformation layer at the beginning of the network structure. This can be done *after* training the network. This layer does not contain learnable weights. This means that after this insertion, we can use PCA-transformed images as inputs to the extended network. While this procedure does not yield the mean dimension on the original data, it allows one to carry out relative comparisons, across alternative network architectures. (See the Appendix for greater details).

We are left to address the first issue mentioned above: The final layer of neural network for classification is usually a softmax layer, leading to a number of outputs equal to the number of classes. The resulting probabilities are natural functions of interest: They need to be sensitive to features in the input that relate to the classes to be predicted and are typically the direct input to the objective function that is used to train the network. While we could analyze the single outputs independently, as is done in Hoyt and Owen (2020a), and also in the layer-by-layer analysis of the work, we would like to have a single number

characterizing the mean dimension of the neural network. When using the cross-entropy for training the neural network, the objective function for a single input-output pair is the negative log-probability of the true class for the input. This quantity depends on all neurons in the last layer before the softmax and is the value being minimized during training. We therefore set the output Y equal to the negative log-probability of the true class.

2.5.1 Where Do Interactions Occur?

In this subsection, we report results for a layer-by-layer analysis of interaction studying a LeNet-5 like architecture (LeCun et al., 2015) trained on the CIFAR-10 image classification data set. We refer to the appendix.

Table 2.1 shows the LAMD of the trained network using the ReLU (mid column) and TanH (right column) activation functions. The left column shows the corresponding layer type to the LAMD. The final average mean dimension of the network with the TanH is larger than with the ReLU activation function. However, looking at the individual steps comparing the LAMD before and after activations, the increase of the LAMD is larger all three times for ReLU than TanH. See for example the increase in LAMD from row 1 to row 2, row 4 to 5 and row 7 to row 8.

2.5.2 When Do Interactions Arise During Training?

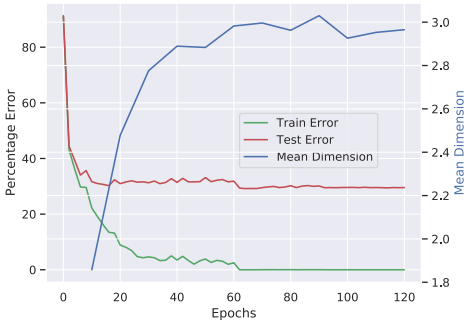
We now focus on evaluating the difference of neural network architectures and on the evolution of the mean dimension during training on CIFAR-10. We consider the following well known neural network architectures: LeNet-5 (LeCun et al., 2015), ResNet-101 (He et al., 2016) and DenseNet-121 (Huang et al., 2017). We refer to the appendix for training details. After training each network for 120 epochs, the estimated mean dimension for LeNet-5, ResNet-101 and DenseNet-121 are 2.97, 7.91 and 5.55. The training errors for

	LAMD	
Layer Type		
Conv2d	0.9991	0.9992
ReLU/TanH	1.2382	1.1842
MaxPool2d	1.0977	1.2396
Conv2d	0.9019	1.1179
ReLU/TanH	1.7292	1.7165
MaxPool2d	3.2240	2.0922
Linear	0.7864	1.1820
ReLU/TanH	2.3429	1.3971
Linear	0.8875	1.1484
NLL	2.6912	3.1786

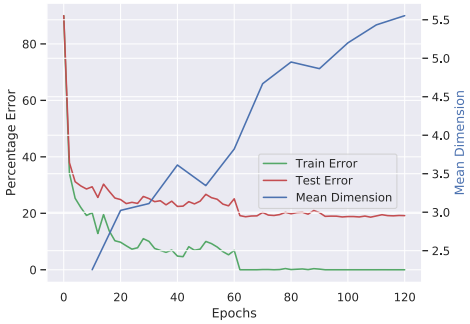
Table 2.1: LAMD of LeNet-5 type architecture trained on CIFAR-10 dataset using ReLU activation function (mid column) and hyperbolic tangent activation function (right column). The left column shows the corresponding layer type to the LAMD, where Conv2d is a 2D convolution layer, MaxPool2D is a 2D max pooling layer, linear is a fully-connected layer and Relu/TanH is the layer where we apply the activation function. NLL is the negative loglikelihood loss.

all three models are zero and test errors for LeNet-5, ResNet-101 and DenseNet-121 are 29.55, 19.18 and 21.58, respectively.

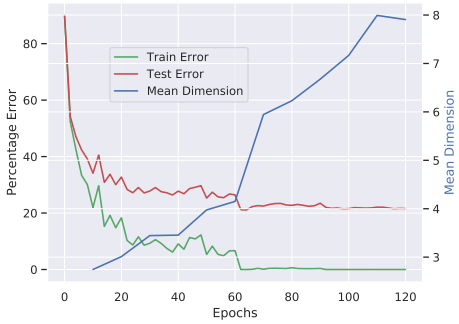
Figure 2.3 displays the mean dimension estimates during training epoch by epoch from 10 to 120 epochs, as well as the train and test errors. We observe that the mean dimension increases as the epochs increase. The increase in mean dimension is steeper for early epochs for the LeNet-5 architecture. The other two architectures do not seem to exhibit this pattern. Deeper networks such as ResNet-101 and DenseNet-121 have a higher mean dimension with respect to the more shallow ones like LeNet-5. This difference between architectures is visible from the beginning of training. We recall that there is one order of magnitude of difference in parameter size between the networks used in our experiment, with ResNet-101 having the largest number of layers followed by DenseNet-121 and then by LeNet-5. On average, the mean dimension is more than 2.5 times higher in ResNet-101 than in LeNet-5 according to our estimates. Thus, the mean dimension seems to be positively correlated with the complexity of the network, with larger networks



(a) LeNet-5



(b) DenseNet-121



(c) ResNet-101

Figure 2.3: Mean dimension (MD, blue), training (green) and test error (red) evolution during training of 120 epochs using different architectures. The mean dimension is estimated every 10^{th} epoch of training. Errors are the percentage of misclassified images in the training and test set respectively.

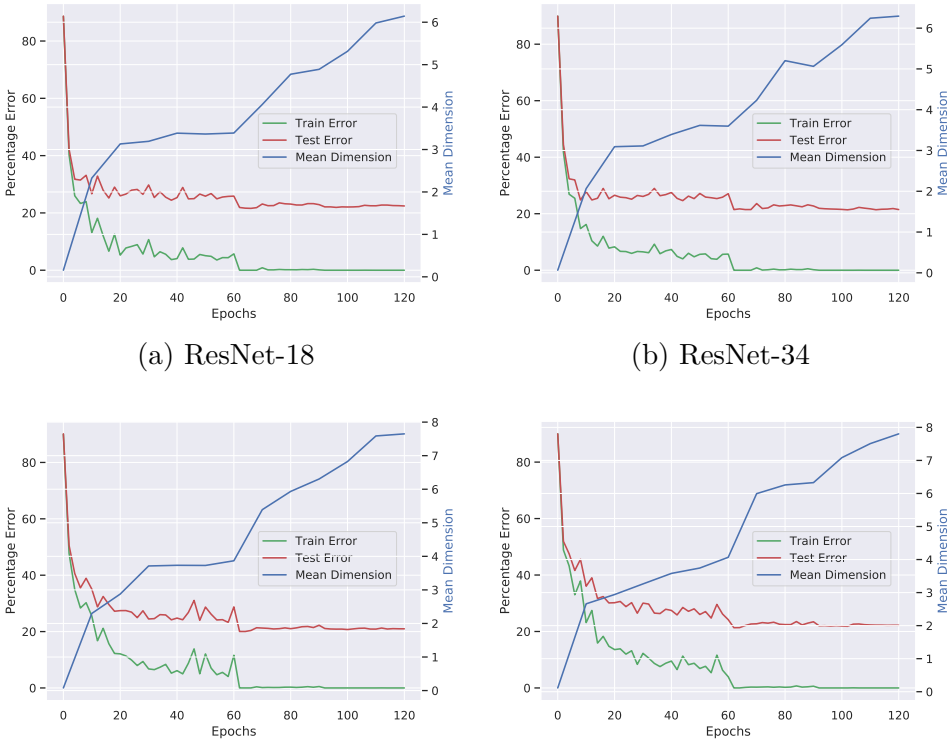
having larger mean dimension.

The graphs in Figure 2.3 may seem to indicate that the mean dimension converges more rapidly in the LeNet-5 model than in the other models. However, this is only due to the truncation at 120 epochs. We performed additional experiments with a higher number of epochs and the final mean dimension does not change substantially as the number of epochs increases also for ResNet-101 and DenseNet-121. That is, the least complex architecture LeNet-5 learns most of the interactions during the first epochs of training, while the other two architectures learn these interactions later on.

It is often true for computer vision tasks that a better accuracy can be achieved with bigger networks and more training, but for some problems these highly complex models might not be useful, for example due to overfitting. The mean dimension might give indications about when this is the case by showing that from a certain complexity on-wards the average interaction size stops increasing. How general this assertion is requires further experiments. These will be part of future research of the authors.

In the next series of experiments, we estimate the mean dimension for the image classification task of CIFAR-10 using different versions of the ResNet architecture. In Fig. 2.4, we show the evolution of the mean dimension as well as training and test error for ResNet-18, ResNet-34, ResNet-50, and ResNet-152 as the number of epochs increases. The corresponding plot for ResNet-101 is in Fig. 2.3. In accordance with the results of the previous experiments, we obtain larger mean dimension estimates for models with a higher number of layers. After 120 epochs these are 6.14, 6.29, 7.65, 8.06 and 7.80, respectively, for ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152.

The analysis of the different depth versions of the ResNet architecture confirms that with increasing number of layers and neurons, a neural network has a higher estimated mean dimension. However, it is interesting to observe that after 50 layers, the mean dimension varies only slightly. The mean dimension of ResNet-101 is slightly higher and the mean dimension of ResNet-152 is even lower than the one of ResNet-101. A



(a) ResNet-18 (b) ResNet-34

(c) ResNet-50 (d) ResNet-152

Figure 2.4: Mean dimension (blue), training (green) and test error (red) evolution during training of 120 epochs using the ResNet architecture with 18 (a), 34 (b), 50 (c) and 152 (d) number of layers. The results using 101 layers (ResNet-101) is in Fig. 2.3. The mean dimension is estimated every 10^{th} epoch of training.

similar behavior is observed in the test error, which is 22.43, 21.47, 20.98, 21.19, 22.20 for ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152, respectively. One might conclude that ever increasing complexity does not lead to ever increasing mean dimension. There seems to be natural bound for the mean dimension using the same architecture. Again this might indicate which complexity could be needed for a certain problem at hand. Another insight could be that once we stop seeing an increase in mean dimension with increasing depth size, we start overfitting and the test error stops decreasing or even starts increasing. This is a natural question for future investigations.

2.6 Discussion

Understanding the structure of a neural network is crucial for increasing interpretability, and understanding interactions plays an important role. However, determining exactly what interactions occur in neural networks used in realistic applications can be infeasible. Relying on the notion of mean dimension, we have proposed a methodology that provides an estimate of the average interaction size in a neural network. We are providing an algorithmic approach that makes it possible to estimate the mean dimension from a given dataset. Moreover we would like to highlight the difference to other methods from the explainability literature such as Layerwise Relevance Propagation (LRP), which is a technique at the individual prediction level, while the mean dimension is based at the dataset and model levels.

We have tested convergence on a test case for which the mean dimension was analytically known. We have then used the mean dimension to study the impact of alternative activation functions, on the synthetic dataset as well as using completely unstructured data.

Our image classification experiments have shown consistently that the mean dimension increases with the complexity of architectures and, within the same architecture, with number of layers. Interestingly, however, from a certain number of layers on-wards the mean dimension stagnates.

Overall, the analyst is equipped with a new tool and algorithmic procedure that allows one to synthesize neural network complexity in a single statistic permitting comparisons of alternative network architectures, activation functions and more.

The work also opens to future research directions. There are of course current limitations connected to, for example, computational power available for our experiments. Repeating the experiments for a larger ensemble of networks and analyzing the mean dimension in relation to generalization, similarly to the idea of Jiang et al. (2019), are avenues of future research.

Appendices

2.A Experimental Details

2.A.1 Inverse-PCA Layer

The network is trained on the original data. This data is then PCA transformed, keeping all components, and used as the input to the neural network after it has been extended with the inverse-PCA layer (see below). We use this extended network for the calculation of the mean dimension, which means that we calculate the mean dimension with respect to the PCA transformed features. The purpose of this additional layer is that we can estimate the mean dimension on uncorrelated features while using a neural network that has been trained with correlated features.

The inverse-PCA layer can be implemented by calculating the matrix of principal components, $\mathbf{c}$, the means of the original features $\bar{\mathbf{x}}^{\mathbf{orig}}$ and the standard deviation of the original features $\sigma^{\mathbf{orig}}$.

The inverse-PCA layer then corresponds to the operation

$$h^1 = (\mathbf{x} * c^T + \bar{\mathbf{x}}^{\mathbf{orig}}) * \sigma^{\mathbf{orig}}, \tag{2.14}$$

where x is an input and h^1 is the output of the inverse PCA layer. Notice that the output of the original neural network applied to an input is equal to the output of the extended neural network when applied to a PCA transformed version of the same input.

This extended neural network is used for calculating the finite differences of Equation (2.11) in the main text. The basis for the calculation of finite differences are pairs of inputs where one feature has been exchanged, see Equation (2.10) and Equation (2.11) of the main text. When combining finite differences with the inverse PCA layer, we exchange these features in the PCA transformed inputs.

Then, to estimate the mean dimension, we can follow the algorithmic procedure introduced in the main paper using the PCA-transformed images as inputs and the extended neural network as model.

Note that the purpose of the additional PCA layer is not dimensionality reduction. We keep the number of dimensions invariant before and after the inverse-PCA layer. The inverse-PCA layer allows us to use inputs that are linearly uncorrelated.

2.A.2 Image Classification

Except where specified otherwise, all neural networks in this section are trained for 120 epochs, with a learning rate of 0.001 using the *Adam* optimizer (Kingma and Ba, 2015). The weights and biases are initialized by the default Kaiming initialization in Pytorch (He et al., 2015).

We train the neural networks on the original images as inputs. We then apply a PCA-transformation to the images and extend the trained neural network with an inverse-PCA layer. We then apply the mean dimension estimation routine on the extended network with the inverse-PCA layer.

The LeNet-5 architecture consists of an input layer, 2 convolutional layers with pooling and a final linear layer. The 10 dimensional output layer with a node for each of the 10 classes is then transformed using a softmax layer and the negative log-likelihood loss serves as a scalar output.

2.A.3 Ishigami Function

We use PyTorch (Paszke et al., 2019) for training the neural network. We generate a dataset consisting of 60,000 instances. One instance is composed of the randomly drawn input features and the Ishigami function evaluation at these inputs as corresponding output. The data is then split into $48,000$ samples for training and $12,000$ samples for

testing. We use 300 nodes in the first layer and 50 nodes in the second one. We train for 100 epochs with the learning rate set to 0.01. Weights and bias are initialized according to Pytorch default Kaiming initializiation (He et al., 2015). We report results for experiments with both the Rectified Linear Unit (ReLU) and the hyperbolic tangent (TanH) activation functions.

2.B Code

The code for reproducing the experiments in the paper is printed in the final appendix of the thesis. The code for training the networks is a modified version of the code accompanying the paper Pittorino et al. (2020). Estimating the mean dimension for LeNet-5, ResNet-101 and DenseNet121 takes 0h:29min, 2h:55min and 6h:49min wall-clock time on a single NVIDIA TITAN RTX. Thus to estimate the mean dimensions using the ResNet-101 architecture for every 10th epoch of a total of 120 training epochs takes 35h on a single NVIDIA TITAN RTX.

References

Apley, D. W. and Zhu, J. (2020). Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 82(4).

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K. R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7):1–44.

Begoli, E., Bhattacharya, T., and Kusnezov, D. (2019). The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence*, 1(1):20–23.

Borgonovo, E. and Rabitti, G. (2020). From Tornado Diagrams to Effective Dimensions. *submitted*.

Breiman, L. (2001). Statistical modeling: The two cultures. *Statistical Science*, 16(3):199–215.

Caflisch, R., Morokoff, W., and Owen, A. (1997). Valuation of mortgage-backed securities using Brownian bridges to reduce effective dimension. *The Journal of Computational Finance*, 1(1).

Cocco, S., Feinauer, C., Figliuzzi, M., Monasson, R., and Weigt, M. (2018). Inverse statistical physics of protein sequences: a key issues review. *Reports on Progress in Physics*, 81(3):032601.

Efron, B. and Stein, C. (1981). The Jackknife Estimate of Variance. *The Annals of Statistics*, 9(3).

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4).

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65.

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018). A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5).

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December.

Homma, T. and Saltelli, A. (1996). Importance measures in global sensitivity analysis of nonlinear models. *Reliability Engineering and System Safety*, 52(1).

Hoyt, C. and Owen, A. B. (2020a). Efficient estimation of the anova mean dimension, with an application to neural net classification. *arXiv preprint arXiv:2007.01281*.

Hoyt, C. R. and Owen, A. B. (2020b). Efficient estimation of the ANOVA mean dimension, with an application to neural net classification. *arXiv*, pages 1–26.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-January.

Ishigami, T. and Homma, T. (1990). An importance quantification technique in uncertainty analysis for computer models. In *[1990] Proceedings. First International Symposium on Uncertainty Modeling and Analysis*, pages 398–403.

Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., and Bengio, S. (2019). Fantastic generalization measures and where to find them.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. *Science Department, University of Toronto, Tech.*

Kucherenko, S., Delpuech, B., Iooss, B., and Tarantola, S. (2015). Application of the control variate technique to estimation of total sensitivity indices. *Reliability Engineering and System Safety*, 134:251–259.

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning.

LeCun, Y. et al. (2015). Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14.

Li, Y. (2017). Deep reinforcement learning: An overview.

Lipton, Z. C. (2018). The Mythos of Model Interpretability. *Queue*, 16(3):31–57.

London, A. J. (2019). Artificial intelligence and black-box medical decisions: accuracy versus explainability. *Hastings Center Report*, 49(1):15–21.

Lundberg, S. M. and Lee, S. I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 2017-Decem.

Montavon, G., Samek, W., and Müller, K. R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing: A Review Journal*, 73:1–15.

Nguyen, H. C., Zecchina, R., and Berg, J. (2017). Inverse statistical problems: from the inverse ising problem to data science. *Advances in Physics*, 66(3):197–261.

Owen, A. B. (2003a). The dimension distribution and quadrature test functions. *Statistica Sinica*, pages 1–17.

Owen, A. B. (2003b). The dimension distribution and quadrature test functions. *Statistica Sinica*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Pittorino, F., Lucibello, C., Feinauer, C., Malatesta, E. M., Perugini, G., Baldassi, C., Negri, M., Demyanenko, E., and Zecchina, R. (2020). Entropic gradient descent algorithms and wide flat minima. *arXiv preprint arXiv:2006.07897*.

Rabitz, H. and Aliş, Ö. F. (1999). General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25(2-3).

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 13-17-Augu, pages 1135–1144.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead.

Sejnowski, T. J. (2020). The unreasonable effectiveness of deep learning in artificial intelligence. *Proceedings of the National Academy of Sciences*, page 201907373.

Sobol', I. (1993). Sensitivity Estimates for Nonlinear Mathematical Models.

Tibshirani, R. (1996). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1).

Tsang, M., Cheng, D., and Liu, Y. (2018). Detecting statistical interactions from neural network weights. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75.

Chapter 3

Using Tools for Interpretability for a Comparison of Neural Network Optimization Routines

3.1 Introduction

There is a growing literature highlighting the need to increase interpretability and explainability of results generated by complex artificial intelligence architectures. New developments and methods are leading to ever improving performances on benchmarks. While the theoretical ground that explains these improvements is not yet settled, it is widely recognized that a key breakthrough in deep learning has been brought by the availability of new and efficient optimization methods.

The standard method of optimizing the loss function in neural networks is stochastic gradient descent (SGD), which uses a low loss as the only metric for evaluating a solution. While leading often to a very good performance, the method may be trapped in sharp local minima, which are less robust towards perturbations of the parameters of the neural network. These sharp minima have been connected to overfitting and there is evidence

that wide, flat minima have better generalization properties overall (Keskar et al., 2016). The idea of seeking flat minima has a long history in machine learning (Hochreiter and Schmidhuber, 1997; Hinton and Van Camp, 1993).

Replicated SGD (rSGD), a so-called *entropic* algorithm, has been proposed as a way to address this problem (Pittorino et al., 2020). The method consists of running standard SGD on several replicas of the same neural network in parallel and adding an interaction term that keeps the parameters of the replicas close to each other. In Pittorino et al. (2020), this interaction is the squared Euclidean distance between the parameters of the replicas and the mean of their parameters. The method can also be seen as an approximation to another entropic algorithm, Entropy-SGD (Chaudhari et al., 2019), which evaluates a point in the space of parameters not only by the loss value associated to this point, but also by the loss values in a region around the point (Baldassi et al., 2016).

Since this method of learning is designed to avoid overfitting, it is interesting to ask how the resulting input-output mapping implied by these solutions differs from the input-output mapping of standard SDG and if such a difference can be interpreted with a view to the different generalization properties of the resulting artificial neural networks.

One line of research is to compare the robustness of the ordinary trained network and the replica trained network by (random) pixel attacks analysis.

We contribute to this research trend by providing a comparison methodology developed in the global sensitivity analysis literature that allow an efficient and thorough comparison of network structures. We use two methods from the interplay of sensitivity analysis and machine learning that answer feature importance and interaction quantification questions. For feature importance we rely on probabilistic sensitivity measures that can be directly estimated from the available dataset. For this task, we frame the analysis in the context of the Xi-method (Borgonovo, E. and Ghidini, V. and Hahn, R. and Plischke, E., 2021). For interaction quantification, we rely on the notion of mean dimension (Hoyt and Owen, 2020; Hahn et al., 2021).

We focus mainly on two questions for models trained on computer vision tasks: Do the two resulting networks rely on different areas in the picture to make their predictions? Do they differ in the size of interactions between pixels that are important for the output?

The remainder of the work is organized as follows. We present related literature in Section 3.2, while the methods are introduced in Section 3.3. In Section 3.4 we present the results, and conclude with a discussion in Section 3.5.

3.2 Related Literature

This section combines the literature from three different research streams. We start by providing the related literature leading to the replica training methodology. Then we present works that focus on feature importance and try to crystallize the important pixels in an image classification task from the explainable artificial intelligence (XAI) and the global sensitivity analysis (GSA) communities. Lastly we introduce the notion of mean dimension and point out its development.

The idea to bias the training towards flat minima has been a topic of interest in machine learning and statistics for several decades (Hochreiter and Schmidhuber, 1997; Hinton and Van Camp, 1993) and recently been investigated for example in their connection to the observation that large-batch methods tend to produce a lower generalization performance than small-batch methods (Keskar et al., 2016). Flatness has also been shown to correlate with generalization in more large-scale studies (Jiang et al., 2019).

One idea that has been explored in this context is to directly optimize a quantity called the *local entropy* (Chaudhari et al., 2019), which is based on the loss value of a complete region instead of a single point and has been conceived in analogy with entropy measures in statistical physics (Baldassi et al., 2015). While several closely related algorithms have been discussed in the literature (e.g. *Elastic Average SGD* (Zhang et al., 2014) and *Parle* (Chaudhari et al., 2017)), we focus in this work on rSGD, which has recently been tested

with success in the deep learning setting (Pittorino et al., 2020). In Pittorino et al. (2020), the authors show an improved generalization performance on several modern architectures and datasets.

Having trained two networks with the two different optimization routines (rSGD and standard SGD), a natural question is whether the networks look at different areas or pixels in an image and if this is connected to a difference in accuracy between the two networks.

Finding the most important areas in images with respect to classification using neural network models is a problem that recently has received a lot of interest and falls into the broader category of feature importance and feature selection methods. We provide here an overview but do not claim to be complete. Works such as Hastie et al. (1994), Boehmke et al. (2020) give a broader review.

When it comes to artificial intelligence applications, as mentioned in Murdoch et al. (2019), techniques can be divided into two main families: individual prediction methods and dataset methods. In the first class of methods are popular techniques such as the Randomized Input Sampling for Explanation (RISE) (Petsiuk et al., 2019), which is a specific algorithm to explain image classifiers. With this technique, a single input image is perturbed by means of randomly generated binary masks, allowing the computation of a score for each pixel, which evaluates the consequent drop in the accuracy and thus is proportional to its importance according to the model. The RISE technique is model agnostic, but it is data aware (it can only be applied to images).

A similar idea is employed by the authors of the Local Interpretable Model-agnostic Explanations (LIME) method (Ribeiro et al., 2016): in this case, the single data input is first preprocessed (i.e. divided into superpixels in the case of images), and then the modified input is perturbed, obscuring one partition of the data at a time; then the consequences on the output of the black-box are measured using a local interpretable model as approximation (e.g. linear regression or tree). LIME is a very flexible technique, since it can be used with any kind of data or black-box.

Recently model-aware techniques on the individual prediction level became more popular. These exploit some characteristics of the respective model structures and so can only be applied to a specific model.

Some of such model-aware techniques for Neural Networks, which make use of the gradient characterisation, are: GradCAM (Selvaraju et al., 2017), Vanilla Gradient (Erhan et al., 2009) and Layerwise Relevance Propagation (LRP) (Bach et al., 2015; Binder et al., 2016). The latter one may provide a very useful tool to explain decisions on an individual image level highlighting the important pixels in an image by propagating importance backward through a neural network.

Notice that all the mentioned techniques create so-called post-hoc explanations (defined in Murdoch et al. (2019)), which provide an understanding of the reasons behind the classification of a certain data instance, after fixing the model structure and parameters.

The comparison of the two networks requires such a post-hoc explanation and hence, we deal with black-box interpretability (as defined in Rudin (2019)), which means that we treat each model as a fixed black-box and we provide explanations without intervening on the architecture of the classifier.

We strive for a "global" comparison of two networks; comparing individual image explanations can be very tedious or even intractable due to the large number of data points needed to train deep neural networks. Therefore we rely on the recently introduced Xi-method of Borgonovo, E. and Ghidini, V. and Hahn, R. and Plischke, E. (2021) for explanation, which is based on probabilistic importance measures from the field of global sensitivity analysis (GSA). The Xi-method is a post-hoc explanation method that works in the context of supervised classification. It is composed of the following ingredients. First, a measure of statistical dependence which is computed on the data (train or test); second a supervised machine learning model. The steps of the method are as follows: the measure of statistical dependence is computed on the original data, without the training of the ML model. Then, the most important pixels according to the measure of statistical

dependence are recorded. Second, an ML model is fitted to the data. If the performance is deemed sufficiently accurate, then the ML algorithm is asked to produce forecast on the same dataset on which the measure of statistical dependence has been computed. Third, the measures of statistical dependence are computed on the forecasts. Fourth, the resulting quantitative results are compared with the results for the measures computed on the data, before the ML model was trained.

Regarding the ingredients of the Xi-method, we recall that in the GSA literature, importance measures were developed to reveal the dependence of a target variable on its inputs. There is a common rationale underlying most of these GSA importance measures (Borgonovo et al., 2016). There are, among others, variance-based sensitivity measures (Iman and Hora, 1990; Saltelli, 2002), sensitivity measures based on the value of information (Oakley, 2009), density-based sensitivity measures (e.g. (Borgonovo, 2007)) and more general distribution based sensitivity measures (e.g. (Baucells and Borgonovo, 2013)). We work with a recent measure of the toolbox of the common rationale that relies on the intuition of exploiting the empirical cumulative distribution function of the model output (Plischke and Borgonovo, 2020). The details are presented in Section 3.3.3.

Our second approach of comparing and analysing the single and the replicated network is based on the mean dimension. The mean dimension is a notion that gives the average interaction size in an input-output pair which has its foundation in the functional Anova expansion (see Rabitz and Aliş (1999) for a review).

The functional Anova decomposition allows to expand a multivariate mapping of the function of interest into 2^d terms, in which second order and higher order terms account for the residual level of interaction (Efron and Stein, 1981; Sobol', 1993). Here, d is the number of arguments of the function.

This representation is exploited in Caflisch et al. (1997) and Owen (2003) for introducing the notion of mean dimension that indicates the average interaction size. Thus, using the notion of mean dimension to compare two networks reveals structural differences between

them. There is recent interest in estimating the mean dimension of neural networks. We recall the work of Hoyt and Owen (2020), where the focus is mainly on different estimation routines of the mean dimension of neural networks, while Hahn et al. (2021) exploits this concept to compare different architectures and show how the mean dimension evolves during training.

3.3 Methods

In this section, we describe the methods used in this work. The optimization methods are described in Section 3.3.1. We introduce the mean dimension and its estimation in Section 3.3.2. In Section 3.3.3, we define global importance measures and outline their estimation. We then describe the modified Xi-method in Section 3.3.4, which provides model explanations on different granularity levels. Lastly, Section 3.3.5 shortly presents correlation metrics, that are used to obtain a quantitative comparison of sensitivity results.

3.3.1 Optimization Methods

In this work, we compare two optimization methods for neural networks, SGD with Adam and rSGD, which we will call single and replicated network, henceforth. We refer to Pittorino et al. (2020) for details on rSGD and review here only the basic idea. While SGD is based on the loss function $\mathcal{L}(\theta)$ for a single neural network with parameters θ, in rSGD the network is replicated y times, which means that we can represent that parameters of the replicated system as the set $\Theta = \{\theta_r\}_{r=1}^{y}$. The loss function $\mathcal{L}_R$ for the replicated system is then defined as

$$\mathcal{L}_R(\Theta) = \sum_{r=1}^{y} \mathcal{L}(\theta_r) + \gamma \sum_{r=1}^{y} |\bar{\theta} - \theta_r|^2, \tag{3.1}$$

where $|\bar{\theta} - \theta_r|^2$ is the squared Euclidean distance between the parameters θ_r for the

replica r and the average parameters over all replicas,

$$\bar{\theta} = \frac{1}{y} \sum_{r=1}^{y} \theta_r. \tag{3.2}$$

While in SGD, one mini-batch of size B is sampled and the gradient of $\mathcal{L}$ is calculated based on this single mini-batch, in rSGD one mini-batch is sampled independently for every replica and the gradient of the replicated loss $\mathcal{L}_R(\Theta)$ is calculated with respect to all replica parameters. The replicas are initialized independently and the interaction parameter γ is increased during training, leading to a collapse of the replicas at the end of training. Then, the average $\bar{\theta}$ can be used as a single neural network for prediction.

The reasoning behind this approach has deep connections to statistical mechanics (Baldassi et al., 2016). It can be derived as an approximation to the *local entropy* loss, which evaluates a single point in parameter space θ by integrating the loss with respect to a second point θ' over the whole parameter space, weighing every point θ' with the distance to θ. In this way, the point θ is evaluated by quantifying how much low-loss volume close to θ is.

The loss in Equation (3.1) can be interpreted in a more intuitive way: The stochasticity inherent in SGD drives the replicas apart, which is countered by the attraction term. This will lead to some typical distance between the replicas for a fixed γ, and it becomes unlikely that all of them get stuck in the same sharp minima at the same time.

3.3.2 Definition of Mean Dimension and Its Estimation

The well-known functional Anova decomposition allows us to expand a multivariate mapping into 2^d terms, where d is the dimension of the feature space. Let $\mathcal{X}$ denote the support of X. Let also $g : \mathcal{X} \to \mathbb{R}$ be a square integrable mapping with respect to $\mathbb{P}_X$. Under the assumption that $\mathbb{P}_X$ is a product measure, in Efron and Stein (1981) it is proved that g can be decomposed in the following expansion:

$$g(x) = \sum_{u \subseteq \{1,2,\ldots,d\}} g_u(x_u), \tag{3.3}$$

where the functional ANOVA effects $g_u(x_u)$ are computed from

$$g_u(x_u) = \int_{X_{\sim u}} g(x) dF_{\sim u}(x_{\sim u}) - \sum_{v \subset u} g_v(x_v). \tag{3.4}$$

The symbol $\sim u$ represents the set of all indices with exclusion of the ones in u.

Building on this decomposition the classical results of Efron and Stein (1981); Sobol' (1993) allow us to decompose the prediction variance as a sum of variance contributions by all subgroups of indices:

$$\sigma^2 = \sum_{u \subseteq \{1,2,\ldots,d\}} \sigma_u^2, \tag{3.5}$$

where the variance contribution of a subgroup u is defined as

$$\sigma_u^2 = \int_{X_u} \left[g_u(\mathbf{x}_u) \right]^2 dF_u(\mathbf{x}_u). \tag{3.6}$$

Then, the mean dimension is defined as (Caflisch et al., 1997; Owen, 2003)

$$D_g = \sum_{u \subseteq \{1,2,\ldots,d\}} |u| \frac{\sigma_u^2}{\sigma^2}, \tag{3.7}$$

where $|u|$ is the cardinality of u.

The terms $\dfrac{\sigma_u^2}{\sigma^2}$ serve as a discrete probability mass function. The last equation can then be interpreted as the average of the cardinality of the subsets of all indices weighted by their fractional contribution to the variance.

For a detailed description of its estimation from a given dataset we refer the reader to Hoyt and Owen (2020); Hahn et al. (2021). We provide a condensed description here. As is shown in Borgonovo and Rabitti (2020), and exploiting results from Owen (2003) , the

mean dimension can be written as

$$D_g = \frac{\sum_{i=1}^{d} \mathbf{V}[\Phi_i(X_i^1)]}{2\sigma^2}, \tag{3.8}$$

where $\mathbf{V}[\Phi_i(X_i^1)]$ is the variance of the finite differences $\Phi_i(X_i^1) = g(X^1) - g(X^0)$. $X^0 = \{X_1^0, X_2^0, \ldots, X_i^0, \ldots, X_d^0\}$ and $X^1 = \{X_1^0, X_2^0, \ldots, X_i^1, ..., X_d^0\}$ are two random points in the input space that differ only in the i^{th} feature dimension.

Then we can estimate the mean dimension from a given data set of inputs x, labels L, sample size N and a neural network trained on this data. Let $x^k = (x_1^k, ..., x_d^k)$ denote the k^{th} input with $k = 1, 2, \ldots, N$. Let also $y^k = g(x^k)$ be the network prediction corresponding to input x^k. We then consider modified samples where a single feature value has been replaced with the same feature value from a different input. We denote such modified inputs as

$$(x_i^l; x_{\sim i}^k) = (x_1^k, \ldots, x_{i-1}^k, x_i^l, x_{i+1}^k, \ldots, x_d^k), \tag{3.9}$$

where the sequence $(x_i^l; x_{\sim i}^k)$ is created by replacing x_i^k with x_i^l in x^k. The finite change with respect to this modification is then defined as

$$\phi_i^{kl} = g(x_i^l; x_{\sim i}^k) - g(x^k). \tag{3.10}$$

Now, suppose we compute r finite differences for each feature i, yielding in a $r \times d$ matrix of finite difference. We then compute the sample variance $\hat{\mathbb{V}}[\phi_i]$ of these finite differences for each feature i. Then, by Equation 3.8, the mean dimension estimate follows as

$$\hat{D} = \sum_{i=1}^{d} \frac{\hat{\mathbf{V}}[\phi_i]}{2\hat{\sigma}^2}, \tag{3.11}$$

where $\hat{\sigma}^2$ is an estimator of the output variance σ^2.

This routine requires independent features. Images, as well as many other real word

data, do not fulfill this requirement. In Hahn et al. (2021), a work-around is suggested. The Principal Component Analysis (PCA) transformation of the images is computed and the transformed images are considered to be the new inputs. Then, the model is extended by a inverse PCA transfromation layer before the first layer. As the name suggests, this layer does not contain learnable weights but performs an inverse PCA transformation of the images. The remaining layers remain unchanged. Thus, the mean dimension can be computed on the space of PCA transformed images, which is sufficient for comparing different neural networks.

3.3.3 Importance Measure from Global Sensitivity Analysis and Its Estimation

Let X and Y be random vector on $(\Omega, \mathcal{B}(\Omega), \mathbb{P})$, with probability measures $\mathbb{P}_Y$ and $\mathbb{P}_X$, respectively. Let $\mathcal{P}$ denote the set of all probability measures on the same probability space and let $d(\cdot,\cdot) : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ denote a distance between distributions. Several global sensitivity measures can be defined through the common rationale (Borgonovo et al., 2016):

$$\xi_X = \mathbb{E}[d(\mathbb{P}_{\mathbb{Y}}, \mathbb{P}_{Y|X})], \tag{3.12}$$

where $\mathbb{P}_{Y|X}$ is the conditional probability measure of Y given X. The intuition at the basis of a probabilistic sensitivity measure is to quantify the degree of statistical dependence between the target Y and one or more covariates (features) X. The symbols $\mathbb{P}_Y$ and $\mathbb{P}_{Y|X}$ in Equation (3.12) have a rather generic meaning. In this work, we consider directly the cumulative distribution function (CDF) F_Y and the conditional CDF $F_{Y|X}$. Thus, the global sensitivity importance measures of i^{th} feature is defined by:

$$\xi_i = \mathbb{E}[d(F_Y, F_{Y|X_i})], \tag{3.13}$$

where F_Y is the marginal CDF of Y and $F_{Y|X_i}$ is the conditional CDF of Y given that we learn X_i; that is $F_{Y|X_i}(y|x) = Pr(Y \leqslant y|X_i = x)$. d is a separation measure defined to measure the distance between two CDFs. The expectation goes over X_i. Different proposals for $d(\cdot, \cdot)$ can be found in the literature.

In this work, we apply the Kolmogorov-Smirnov distance between two CDFs, $d_i^{KS} = \sup_y |F_Y(y) - F_{Y|X_i}(y)|$. Therefore, the reference probabilistic sensitivity measure becomes

$$\xi_i^{KS} = \mathbb{E}[\sup_y |F_Y(y) - F_{Y|X_i}(y)|]. \tag{3.14}$$

We omit the superscript 'KS' in the following. This sensitivity measure is transformation invariant, and is readily estimated from the knowledge of the empirical cumulative distribution functions (Borgonovo et al., 2014).

For detailed information of the estimation routine, we refer to Plischke and Borgonovo (2020) and provide a concise description here. In the case of X_i being continuous, the routine requires to define a partition. The key intuition lies in replacing the point-conditional CDF $F_{Y|X_i=x}$ with the class-conditional CDF $F_{Y|X_i \in \mathcal{X}_i^k}$. Let $\mathcal{X}_i$ denote the support of feature X_i, $i = 1, 2, ..., n_X$, where n_X is the feature dimension. Let also $\mathcal{K}_i = \{\mathcal{X}_i^1, \mathcal{X}_i^2, \ldots, \mathcal{X}_i^K\}$ denote a partition of $\mathcal{X}_i$, i.e., a finite or countable collection of subsets of $\mathcal{X}_i$ such that $\mathcal{X}_i = \cup_{k=1}^K \mathcal{X}_i^k$ and such that $\mathcal{X}_i^k \cap \mathcal{X}_i^j = \varnothing$, for $k \neq j$ and $i = 1, 2, ..., n_X$.

In the case of X_i being a discrete variable, then the partition is immediately given by the discrete set of realizations of X_i.

For the sensitivity measure in Equation (3.13), the following expression represents the equation of a given data estimator given partition $\mathcal{K}_i$ (Plischke and Borgonovo, 2020):

$$\hat{\xi}_i(\mathcal{K}_i) = \sum_{k=1}^{K} \widehat{Pr}(X_i \in \mathcal{X}_i^k) d(\hat{F}_Y, \hat{F}_{Y|X_i \in \mathcal{X}_i^k}), \tag{3.15}$$

where $F_{Y|X_i \in \mathcal{X}_i^k}(z) = Pr(Y < z|X_i \in \mathcal{X}_i^k)$ denotes the conditional CDF of Y given $X_i \in \mathcal{X}_i^k$ for $i = 1, 2, ..., n_X$. $\hat{F}_Y$ and $\hat{F}_{Y|X_i \in \mathcal{X}_i^k}$ are the estimators of F_Y and $F_{Y|X_i \in \mathcal{X}_i^k}$. Obtaining

consistent CDF estimators is the first step for calculating the estimator in Equation (3.15). A natural candidate for this purpose is the empirical CDF of Y. Given n realizations $(y_j), j = 1, 2, ..., n$ of the random variable Y, the empirical CDF of Y evaluated at y is defined by counting the number of realizations below or equal to y and dividing by the sample size,

$$\hat{F}_Y(y) = \frac{1}{n}\#\{y_j : y_j \leqslant y\}, \tag{3.16}$$

where #A denotes the number of elements in the set A. The estimation of the conditional CDF follows analogously.

$\widehat{Pr}(X_i \in \mathcal{X}_i^k)$ is the estimated probability that X_i falls into $\mathcal{X}_i^k$. Let us denote as n_i^k the number of realizations where the i^{th} feature X_i falls in $\mathcal{X}_i^k$, then we get the estimated probability by dividing this count by the number of all realizations n; $\widehat{Pr}(X_i \in \mathcal{X}_i^k) = \frac{n_i^k}{n}$.

Hence we have:

$$\hat{\xi}_i(\mathcal{K}_i) = \sum_{k=1}^{K} \frac{n_i^k}{n} d(\hat{F}_Y, \hat{F}_{Y|X_i \in \mathcal{X}_i^k}). \tag{3.17}$$

3.3.4 The Xi-Method

In this section we describe a modified version of the Xi-method introduced in Borgonovo, E. and Ghidini, V. and Hahn, R. and Plischke, E. (2021). The modified Xi-method provides explanations of a trained model in a classification setting with n_L different target labels $\ell_1, \ldots, \ell_{n_L}$. Specifically, the modified Xi-method requires the definition of an appropriate scalar output representation of the classification problem. The definition of the modified Xi-method follows in Section 3.3.4.1 and the estimation aspects in Section 3.3.4.2.

3.3.4.1 Definition

The Xi-method applies the probabilistic sensitivity measures that were previously defined to obtain post-hoc explanations for a model at different granularity levels. This is achieved by applying the importance measures not to the true target level but to the model output

and by that revealing the dependencies of the trained model on the inputs. It provides insight into the network's decision on a dataset and at a target class level respectively described in Paragraphs a) and b). Furthermore it can highlight differences between misclassified and correctly classified images, see Paragraph c), and a combination of the mentioned granularities.

In the case of a classification problem, we need to define a scalar model output . We consider the negative loglikelihood loss (NLL) Z, that is the negative loglikelihood of the correct class probability, as output. Other choices are possible, such as the predicted label or even all class probabilities that result from the softmax layer. With the choice of the NLL as output, the Xi-method applies to the importance measures using CDFs for Equation (3.13).

a) Explanation at the dataset level Based on Equation (3.12), we can define the explanation at a dataset level based on the separation measure $d(\cdot, \cdot)$ of a ML model whose output is

$$\xi_X^Z = \mathbb{E}[d(\mathbb{F}_Z, \mathbb{F}_{Z|X})].$$

Observe that ξ_X^Z depends on the separation measure $d(\cdot, \cdot)$: in this way, we can obtain explanations inheriting different properties from $d(\cdot, \cdot)$.

The quantity defined above is particularly useful for the purpose of understanding the most important covariates for the general prediction task. But sometimes we may need to retrieve the most important features for a singular response class, or perhaps to visualize what the machine learning model is actually seeing in unstructured data such as images, in order to predict a specific target label. For this, we resort to explanations at a class level.

b) Explanation at a class level In this case, we may be interested in retrieving an explanation for each target class, answering the question *What is important for the model*

when it evaluates inputs from class ℓ_r?

To this aim, we consider only inputs falling into the class of the target label ℓ_r. Then, the explanation based on the separation measure $d(\cdot,\cdot)$ for the target class ℓ_r will be given by

$$\xi_X^r = \mathbb{E}[d(\mathbb{F}_Z^r, \mathbb{F}_{Z|X}^r)],$$

where $\mathbb{F}_Z^r$ is the CDF of Z on inputs with label ℓ_r and $\mathbb{F}_{Z|X}^r$ is conditional CDF of Z given X on the same inputs.

c) Explanation depending on the "correctness" of the prediction Another inside is provided by a further level of granularity. The GSA importance measures can be computed on either correctly classified or misclassified inputs. We denote the importance measures on correctly classified inputs as

$$\xi_X^c = \mathbb{E}[d(F_Z^c, F_{Z|X}^c)],$$

and on misclassified images as

$$\xi_X^w = \mathbb{E}[d(F_Z^w, F_{Z|X}^w)].$$

Here F_Z^c is the CDF of Z on inputs that were correctly classified and $F_{Z|X}^c$ is the conditional CDF of Z given X on the same set of inputs. Analogously for the CDFs in the misclassified case.

d) Combination a), b) and c) A natural extension of Paragraphs b) and c) is a combination of the respective granularities. Thus we denote the GSA importance measures of correctly classified images with label ℓ_r as

$$\xi_X^{c,r} = \mathbb{E}[d(F_Z^{c,r}, F_{Z|X}^{c,r})],$$

and misclassified images with label ℓ_r as

$$\xi_X^{w,r} = \mathbb{E}[d(F_Z^{w,r}, F_{Z|X}^{w,r})],$$

where $F_Z^{c,r}$ is the CDF of Z on inputs of class ℓ_r that were correctly classified and $F_{Z|X}^{c,r}$ is the conditional CDF of Z given X on the same set of inputs. Analogously for the CDF in the misclassified case.

3.3.4.2 Estimation

The estimation of the modified Xi-method follows the estimation routine of GSA importance measures from Section 3.3.3. The data used to estimate the empirical CDFs now varies with the choice of the granularity of the analysis. The estimation of the Xi-method explanation then follows the estimation routine of GSA importance measures in Section 3.3.3 using as data a subdataset, which depends on the chosen granularity level. For the estimation of the Xi-method explanation of the full data (Paragraph a) in Section 3.3.4.1), we apply the estimation routine to the full dataset D. Estimating the Xi-method explanations from inputs with target label ℓ_r (Paragraph b) in Section 3.3.4.1) requires filtering the dataset to inputs with label ℓ_r. This subdataset is called D^r and is then used to estimate the importance measures. Similarly, for the granularity level that separates correctly classified and misclassified input, the dataset is split into correctly classified inputs and misclassified ones. We denote these subsets D^c and D^w, respectively. The filtered dataset of a combination of the target level and the correctness is denoted as $D^{c,r}$ in the case of correctly classified inputs from class ℓ_r and $D^{w,r}$ in the case of misclassified inputs from class ℓ_r.

3.3.5 Correlation Metrics

We are comparing quantitatively the explanations from the modified Xi-method of the single and the replicated network using some correlation metrics. This section provides a concise introduction of these metrics. The first one is the very well-known Spearman's rank correlation. Second, we compute the Pearson correlation of the Savage scores of the importance measures (Iman and Conover, 1987). The Savage scores correlation does not compute the correlation between the ordinary rank orders, but of quantities related to the rank that give more weight to influential features. The reasoning behind this choice is that the exact order of the less important pixels in the ordering of the most important features is decisive for understanding what drives a network decision, while the exact order of the less important features is subjacent. Let r_i be the rank of feature i. Then, the Savage score of feature i is defined as $s_i = \sum_{j=r_i}^{d} \frac{1}{j}$, where d is the total number of features. The correlation of Savage scores is a concordance measure that is more sensitive to agreement on the top rankings.

3.4 Experiments and Results

In this section we present our results of analysing the differences between the single and the replicated network using explanations of the Xi-method (Section 3.4.2) and the notion of mean dimension (Section 3.4.1). We train the well-known convolutional neural network architectures on the CIFAR-10 database (Krizhevsky, 2009). CIFAR-10 is a dataset of natural images such as airplanes and horses. It consists of 6×10^4 color images with 32x32 pixels and 3 color channels. The images are evenly divided into 10 classes.

We use 10^4 images for testing and the rest for training. We train the models using both SGD and rSGD with 3 replicas. These models will be referred to as *single network* and *replicated network*, henceforth.

3.4.1 Comparison Using Mean Dimension

The average interaction size is a distinguishing property of neural networks. We exploit the notion of mean dimension to compare networks trained by the standard approach with networks having the same architecture design trained by the replicated optimization method. We are comparing three different network architectures: LeNet-5, DenseNet-121 and ResNet-101.

The LeNet architecture (LeCun et al., 2015) is a 7 layer neural network consisting of the input layer, two convolutional layers followed by pooling layers and another convolutional layer followed by the output layer. ResNet-101 (He et al., 2016) and DenseNet-121 (Huang et al., 2017) are two deep convolutional neural networks. The ResNet-101 architecture comprises 101 layers featuring residual connections. It has about 42 million learnable parameters. In the DenseNet-121 architecture, each layer uses the feature maps of all preceding layers as inputs. All models in this section are trained for 120 epochs. The learning rate is set to 0.001 and *Adam* is used as optimizer (Kingma and Ba, 2015). The weights and biases are initialized by the default Kaiming initialization in Pytorch (He et al., 2015).

LeNet-5 returns an test error of 26.68% for the replicated network and 29.54% for the single network. The train error of the replicated is almost zero. For all other trained networks, the train error is exactly zero. The test error of the ResNet-101 architecture is 19.23% for the replicated network and 21.58% for the single network. While DenseNet-121 returns an test error of 17.20% for the replicated network and 19.17% for the single one. We are aware that there are networks achieving better accuracy on the CIFAR-10 data, but for our purposes it is already good enough.

For each network architecture we compare the models resulting from the two different optimization approaches. We estimate the mean dimension of the different pairs of neural networks using the estimators from Equation (3.11). Since pixels of images are highly

correlated, we work on the space of PCA-transformed images and extend the network by adding an inverse PCA-layer as described in Section 3.3.2.

The routine requires the definition of a scalar output. We consider the negative loglikelihood loss as a natural choice in this setting.

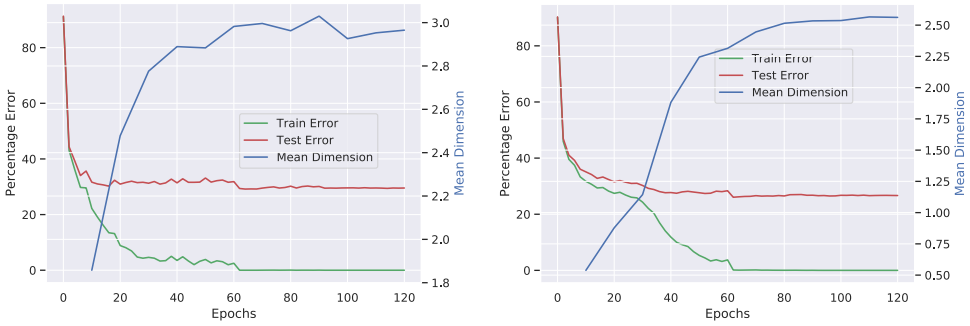
Let us start with analyzing the average interaction size for the LeNet-5 architecture. The single network has an estimated mean dimension of about 2.97, while the replicated network has a mean dimension of 2.56. We have replicated this experiment several times using the same architecture on CIFAR-10 and the results remained the same up to a deviation of 0.1. Thus, this difference between the single and replicated method is coherent and does not depend on the random initialization of the networks.

The model using the ResNet-101 architecture has a mean dimension of 7.91 for the single network and 8.55 for the replicated network. While the estimation procedure using the DenseNet-121 architecture yields a mean dimension of 5.55 for the single network and 7.24 for the replicated network.

Though the notion of mean dimension clearly shows a difference between the models, there does not seem to be a systematic relationship: a single network does not necessarily have a smaller mean dimension than the replicated network. The single network using LeNet-5 architecture has a higher mean dimension than the replicated one, while for the more complex architectures, ResNet-101 and DenseNet-121, this relationship is reversed. However, this result suggests that these models differ in terms of the average interaction size.

Since the two optimization methodologies basically change the way the two networks move through the parameter space during training, it is of interest to analyse the development of the mean dimension with increasing number of epochs.

Fig. 1 shows such a development of the mean dimension for the single and the replicated network using the LeNet-5 architecture. We also plot the training and test error behavior during training in green and blue, respectively. The errors are calculated every second



(a) LeNet-5, single (b) LeNet-5, replicated

Figure 1: Mean dimension (MD, blue), training (green) and test error (red) evolution during training of 120 epochs once with the standard approach and once with replicated method using LeNet-5 architecture. The mean dimension is estimated every 10^{th} epoch of training. Errors are the percentage of misclassified images in the training and test set respectively.

epoch.

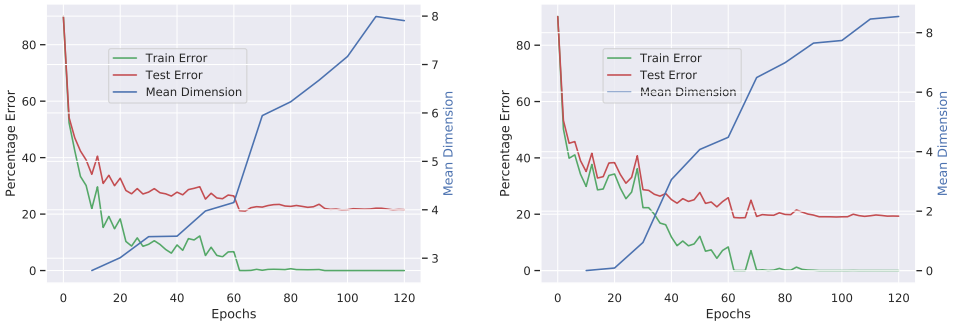
The overall behavior during training is similar for the two networks but the mean dimension increases more smoothly for the replicated network. Fig. 2 and Fig. 3 show the same analysis for the ResNet-101 and the DenseNet-121 architectures comparing the two optimization algorithms. The overall picture is again quite similar between the single network and the replicated network. But again for both architectures, the mean dimension of the replicated network increases more smoothly.

3.4.2 Comparison Using Modified Xi-Method Explanations

We now apply the modified Xi-method introduced in Section 3.3.4. The prediction C of a well trained neural network should show dependence on features X_i that are important for the network's decision. In an image classification task, the Xi-method explanations ξ_i should reflect the dependence of the network's decision on the pixels.

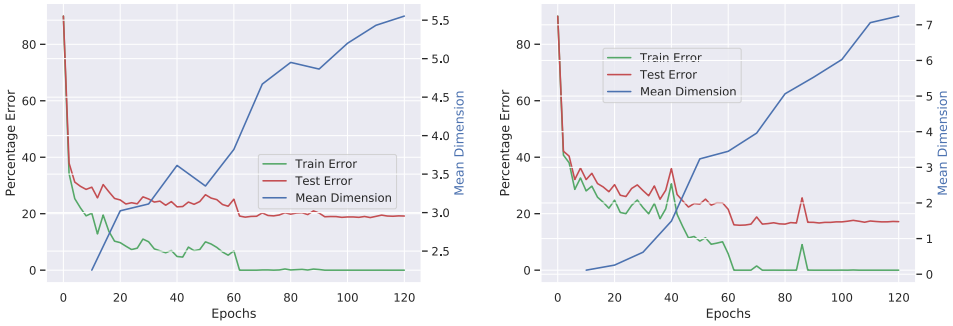
Using the CIFAR-10 database, we get 3,072 explanations from the modified Xi-method, one for each pixel indicating its importance to the neural network.

We train a single network and a replicated network for 60 epochs using the LeNet



(a) ResNet-101, single (b) ResNet-101, replicated

Figure 2: Mean dimension (MD, blue), training (green) and test error (red) evolution during training of 120 epochs once with the standard approach and once with replicated method using ResNet-101 architecture. The mean dimension is estimated every 10^{th} epoch of training. Errors are the percentage of misclassified images in the training and test set respectively.



(a) DenseNet-121, single (b) DenseNet-121, replicated

Figure 3: Mean dimension (MD, blue), training (green) and test error (red) evolution during training of 120 epochs once with the standard approach and once with replicated method using DenseNet-121 architecture. The mean dimension is estimated every 10^{th} epoch of training. Errors are the percentage of misclassified images in the training and test set respectively.

architecture. For both networks, the training errors reach zero. Errors are the percentages of misclassified images. The final error on the test set is 27.73% for the single network and 25.34% for the replicated network.

One major advantage of the explanation provided by the modified Xi-method is the ample opportunities it offers to gain interpretability insights. Following the modified Xi methodology presented in Section 3.3.4, we can gain interpretability on different granularity levels by filtering the dataset accordingly. Either the data is filtered for a specific class, for misclassified or correctly classfied images or a combination of the two. We then progress with calculating modified xi explanation on the filtered dataset.

We mainly focus on explanations on the training data, since the smaller test dataset may weaken the validity of the estimation procedure. We also show results for the test set, however. In this particular setting of zero training error, the analysis of misclassified images is omitted for the training set analysis, but can still provide meaningful insight in other settings.

The modified Xi-method explanations are visualized as heatmaps in the following section and for a more quantitative comparison of the networks, we also give two correlation metrics introduced in Section 3.3.5, computed between the explanations of the single and the replicated neural network.

3.4.2.1 Heatmaps and Rankcorrelation of Importance Measures

One problem that arises with such a high feature dimensionality, as it is usual in models using images, is the illustration of the feature importance explanations. We decide to use heatmaps to highlight the different regions where the two networks 'look at'. In the particular case of color images with three channels, we elaborate two ways of plotting heatmaps:

1. For each color channel (red, green, blue), we provide an individual plot. These heatmaps show the explanation value of the pixels of that channel (note that the

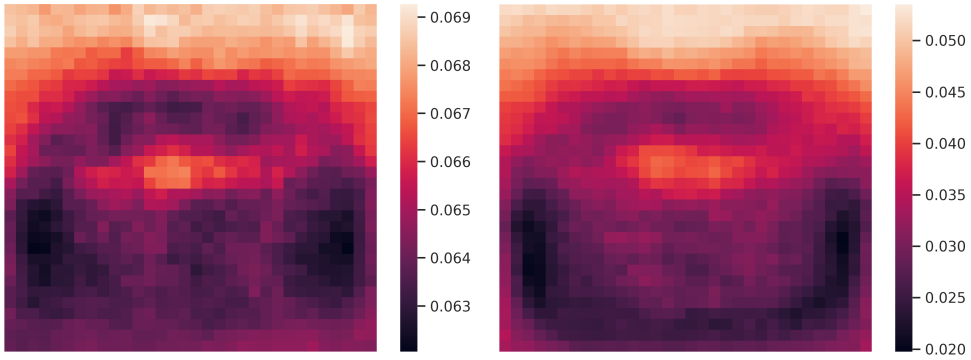


Figure 4: Aggregated heatmap of Xi-method explanations using all images. (left): single network. (right): replicated network

estimation of the modified Xi-method was done on all pixels at the same time). This approach is called *channel heatmap*, henceforth.

2. We provide one aggregated heatmap for the image, which aggregates explanations of the three color channels at the same position. Hence, this yields an explanation for the position of the pixel in the image, by taking the mean of the channel importance values. This approach is called *aggregated heatmap*, henceforth.

We start with the most general case considering the full dataset, training and test data combined. Fig. 4 shows the heatmaps of the Xi-method explanations for the single and the replicated networks.

Though the overall highlights seem to be quite similar, one can clearly see differences between the two plots. The transitions of neighboring Xi-method explanations seem to be more smoothly for the replicated network. For both networks the centre and the upper part are the most important regions of the image. However, there is an important difference. The heatmaps place importance measures on a relative scale. The scales in the heatmaps of the left and the right panel in Fig. 4 are different. The importance measures in the left panel range from ~ 0.062 to ~ 0.070; the ones in the right panel from ~ 0.02 to ~ 0.055. Thus, the replicated network seems to have a more distributed view of the image.

This visual impression of dispersion is also confirmed by the wider range of importance

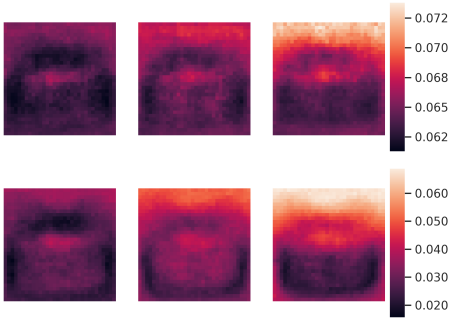


Figure 5: Channel heatmap of Xi-method explanations using all images. (above): single network. (below): replicated network

measures of the replicated model compared to the single model. The importance measures of the latter one vary between ~ 0.062 and ~ 0.070, while the range of the replicated model lies between ≈ 0.02 and ≈ 0.055.

The comparison between the single and the replicated model for each of the three color channels are shown in Fig. 5. The ranges of the scale in the two legends display the importance measures referring to individual pixels (and not the aggregated numbers as in Fig. 4). Again, the visual impression of dispersion is also confirmed by the wider range of importance measures of the replicated model compared to the single model. The importance measures of the latter one vary between ~ 0.062 and ~ 0.070, while the range of the replicated model lies between ~ 0.02 and ~ 0.06.

It is noticeable, that for both networks the upper part of the image, in particular the green (2nd channel) and the blue (3rd channel) color channels are important. This might indicate that the background plays an important role in classifying images for both networks, which is not a very desirable property. Again, especially for the third channel, it seems that the robust network looks more at the whole image.

Thus, one can visibly confirm that the two networks find different minima in the loss landscape.

We now compare the rankings of the importance measure using the Spearman's rank correlation and the Savage score correlation. The correlation results corroborate the

impressions from the visual analysis of the heatmaps. The Spearman's rank correlation gives 0.907 and the Savage score correlation 0.943, indicating that there is in particular a high agreement on the most important pixels.

We now analyse the difference between the two networks considering only correctly classified or misclassified images. We should note here that misclassfied images all fall into the test set, since we have zero training error. The heatmaps of correctly classified images resemble very much the ones using all images. Fig. 6 and Fig. 7 show the sum and the channel heatmap respectively for all correctly classified images. The results confirm the previous findings of the wider range of the importance measures of the replicated model compared to the single model and slightly different areas of importance. Again the difference in heatmaps generated by the importance measures for models using different optimization methods is clearly visible. The correlation metrics are as well quite similar to the ones considering all images; the Spearman's rank correlation and the Savage score yield 0.891 and 0.927.

Fig. 8 and Fig. 9 show the heatmaps for all misclassified images. Interestingly, in contrast to the previous heatmaps, the importance measures do not show any form of a structure. The heatmaps seem to be random noise and the importance measures computed on these subdataset are more scattered through the whole image. Furthermore, the range of the importance measures are now more similar than considering only correctly classified images or the full data. This might have two reasons: the smaller sample size to estimate the explanation values and the fact that all samples are from the test set.

As expected, the correlation metrics are quite low for the misclassified images, yielding 0.152 and 0.195 for the Spearman's rank correlation and the Savage score correlation, respectively.

Hence, to analyse if the source of the unstructured heatmaps of misclassified images lies in the fact that they are misclassified or that they stem from from the test set, the explanations of the modified Xi-method of the images from only the test set might provide

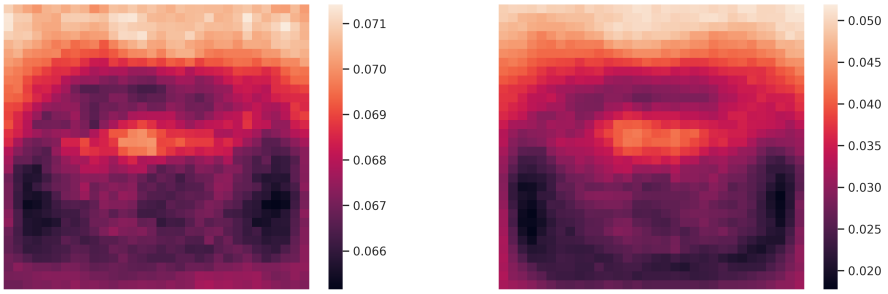


Figure 6: Aggregated heatmap of Xi-method explanations for correctly classified images. (left): single. (right): replicated

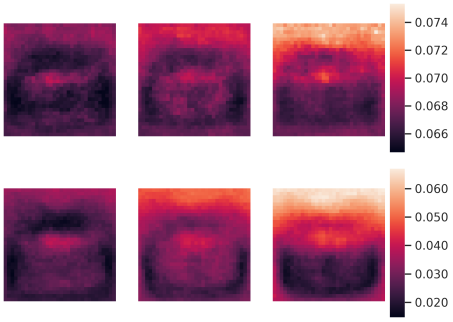


Figure 7: Channel heatmap of Xi-method explanations for correctly classified images. (above): single. (below): replicated

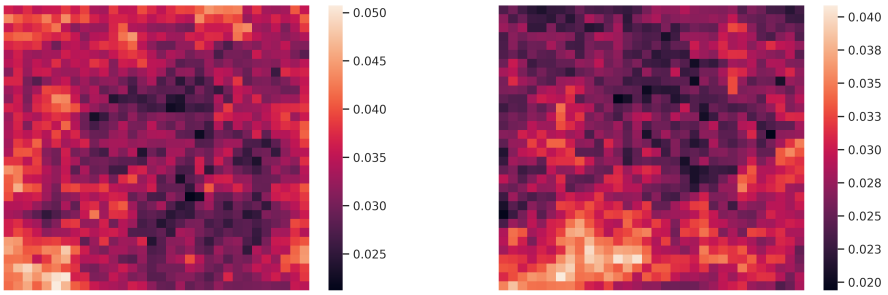


Figure 8: Aggregated heatmap of Xi-method explanations for misclassified images. (left): single. (right): replicated

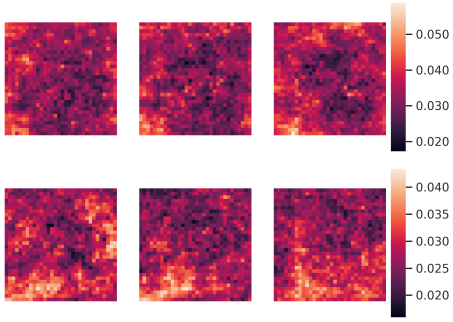


Figure 9: Channel heatmap of Xi-method explanations for misclassified images. (above): single. (below): replicated

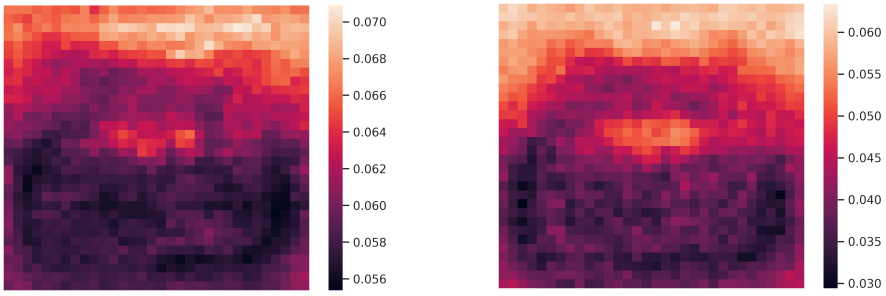


Figure 10: Aggregated heatmap of Xi-method explanations for images in the test set. (left): single. (right): replicated

an answer.

Fig. 10 and 11 show the aggregated and channel heatmaps using only the test set, but the heatmaps seem to be quite similar to the ones considering all data. Hence, the reason for the unstructured heatmaps of misclassified images lies solely in the fact that they are misclassified or in the small sample size, which is only around 1/4 of all images in the test set.

Another interesting insight could be gained by looking at the difference in heatmaps filtering the inputs to only specific classes. An interesting observation emerges for the subdataset with label *horse*. Fig 12 shows the aggregated heatmaps for these images using the full dataset: For the single network almost all areas of the image are equally (un-)important and only a few pixels in the lower center as well as in the lower border seem to stand out. For the replicated model we have some very bright areas in the center

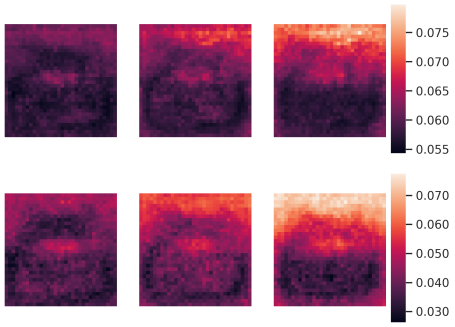


Figure 11: Channel heatmap of Xi-method explanations for all images in the test set. (above): single. (below): replicated

and also the lower and upper end of the image highlighting their relative importance. This could lead to the interpretation that especially for the single network pixels relating to the ground are more relevant than pixels relating to the object itself, e.g. the green pasture on which the horse grazes. The channel heatmap in Fig. 13 strengthens the presumption, by showing that important pixels are at the bottom of the green channel. The other areas and also the other channels are indicated as relatively equally (un-)important with the exception of the lower end of the red channel where some relative importance is found. These comparisons also illustrate a large difference between the single network and the replicated network. The single network seems to make its decision based on the ground while in the replicated network, the human eye may even recognize the contours of a horse.

Also, unsurprisingly, the Spearman and the Savage score correlations between explanations from the single network and the replicated one are comparably low with 0.294 and 0.418. Compared to the results from the importance measures considering all correct classified images, these numbers show that indeed there is a low correlation among the explanations from the two networks. This analysis of the class *horse* reveals that the two networks have indeed learned different mappings.

The heatmaps of considering only correctly classified horses are very close to the results taking into account all images with label horse, see Fig. 14. Conversely, considering only misclassified images with true label horse, the results resemble very much the ones of all

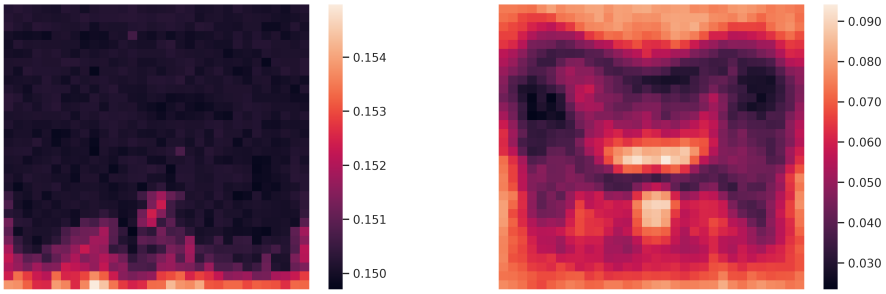


Figure 12: Aggregated heatmap of Xi-method explanations for images with true label horse. (left): single. (right): replicated

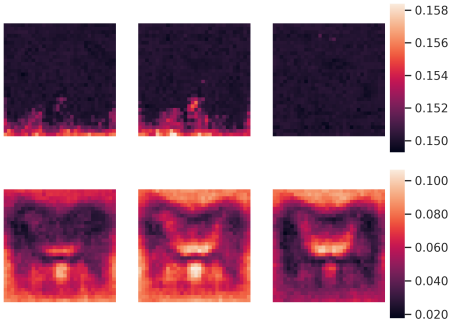


Figure 13: Channel heatmap of Xi-method explanations for images with true label horse. (above): single. (below): replicated

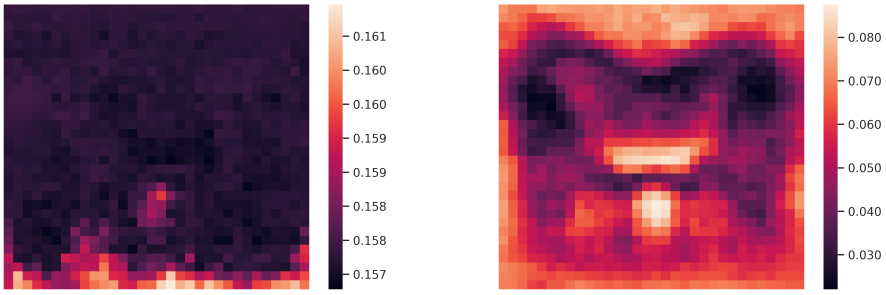


Figure 14: Aggregated heatmap of Xi-method explanations for correctly to class horse classified images. (left): single. (right): replicated

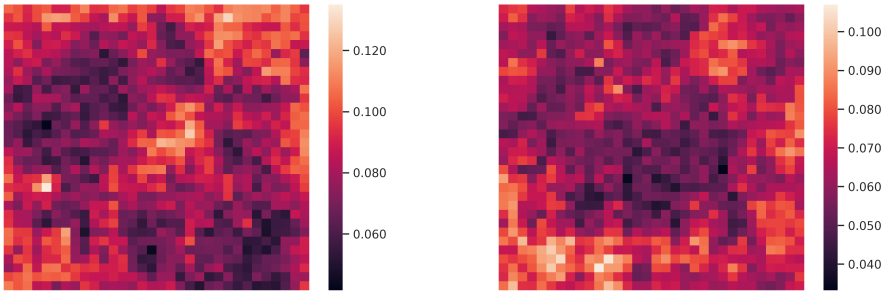


Figure 15: Aggregated heatmap of Xi-method explanations for wrongly to class horse classified images. (left): single. (right): replicated

misclassified images as shown by the scattered pixel importance throughout the heatmaps in Fig. 15. Again caution is advised due to the small sample size of the misclassified images with label *horse*.

These examples show that the modified Xi-method provides a tool for not only analyzing differences in important areas of images but also for detecting possible biases and spurious explanations that neural networks might learn. Further examples and heatmaps are added in the appendix.

3.5 Discussion

We have compared convolutional neural networks trained on an image classification task using two different optimization approaches, namely SGD and rSGD. We have exploited the notion of mean dimension and introduced a new explanation tool, the modified Xi-method, which is based on probabilistic sensitivity measures.

The notion of mean dimension, which provides an average interaction size, is a model property that might uncover dissimilarities between networks trained with the two different optimization approaches. We have run our experiments with the LeNet-5, ResNet-101 and DenseNet-121 architectures and for each architecture we have obtained different mean dimensions for the single and the replicated network. Though these differences are not coherent with the choice of the optimization approach, meaning that the replicated network

does not always have a higher mean dimension than the single network, this result still shows that the networks learn different inner representations. Furthermore, the evolution of the parameters in the loss landscape using the rSGD approach, which is specifically designed to search for flat minima, seems to be reflected in the smoother increase of the mean dimension during training.

The modified Xi-method is a tool for highlighting relevant regions in an image classification task. It is a model agnostic explanation method and can be used to increase interpretibility of any black-box model. The explanations from the modified Xi-method are visualized using heatmaps and clearly highlight differences in important areas for the two networks. The replicated network distinguishes more between important and less important pixels.

One major advantage of this tool is the granularity available to gain interpretability. The Xi-method indicates a potential bias in the trained single network when analysing the explanation for the class *horse*. The explanations show that the single network learns to focus on the ground where the horse is standing instead of the horse itself. While the explanations of the replicated network are more dispersed throughout the image and even the contours of a horse are identifiable in the heatmap. The modified Xi-method provides a quick model agnostic tool for black-box model insights and can be used in general settings, not only for comparison of networks optimized with alternative methods. Hence, although the accuracy of the models trained using the SGD method and the rSGD method are quite similar, we show that these networks indeed have different properties such as average interaction size and focus on different parts of the image.

Future research includes applying this method to different types of black-box models and other tasks besides image classification as well as a deeper comparison with existing tools used for interpretability of neural networks. An extension is also to deploy importance measures that consider the multivariate nature of a network's predictions, i.e., the fact that the network produces not only a most likely label, but a vector of probabilities for

each class. Here, one might assign a Dirichlet prior on this vector. Then, instead of estimating the empirical CDF and the conditional empirical CDF, one fits a Dirichlet distribution and a conditional Dirichlet distribution to the network predictions. Such an explanation would take into account the full output vector. An appropriate separation measure between Dirichlet distributions could be the Kullback-Laibler divergence. Such an explanation would have incorporated the full output vector. An appropriate separation measure between Dirichlet distributions could be the Kullback-Laibler divergence.

Appendices

3.A Estimation Details

As the choice of the partition size K in estimating importance measures from GSA in Section 3.3.3 is important, we will give the details to the experiments in Table 3.A.1.

Xi-method explanations for ...	**Partition size K**
all images	15
correctly classified images	10
misclassified images	5
images from one specific class	6
correctly classified images from one specific class	5
misclassified images from one specific class	4

Table 3.A.1: Partition sizes K for the Xi-method explanation estimation.

3.B Further Heatmaps

We are presenting here further Xi-method explanations of specific classes that were not shown in the main text in order to not overload Section 3.4.2.

We start with the class *automobile*. Fig. 3.B.1 and Fig. 3.B.2 show the aggregated and channel heatmaps for this class. The single network has a concentrated focus on

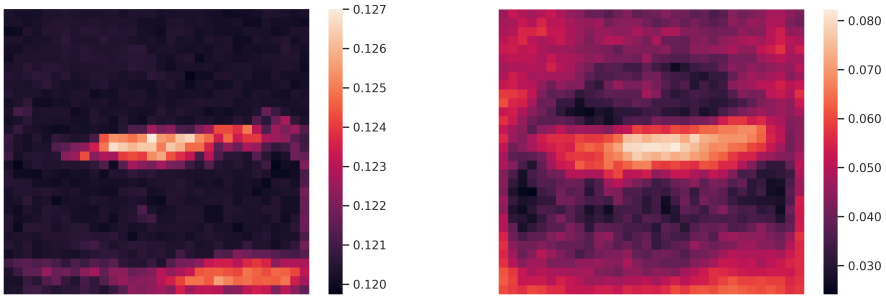


Figure 3.B.1: Aggregated heatmap of Xi-method explanations for images with true label automobile. (left): single. (right): replicated

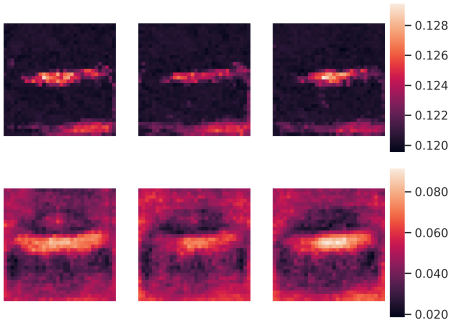


Figure 3.B.2: Channel heatmap of Xi-method explanations for images with true label automobile. (above): single. (below): replicated

the center and the ground of the image while the replicated one has its important pixels more dispersed throughout the image. The different color channels do not have a specific influence as the similar heatmaps for each channel show.

Furthermore, the heatmaps for the class *deer* are shown in Fig. 3.B.3 and Fig. 3.B.4. Differently from the explanations for the class *horse*, the explanations here are quite similar for the two networks. Interestingly for this class, both networks seem to have learned more the background than the center of the image.

References

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K. R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7):1–44.

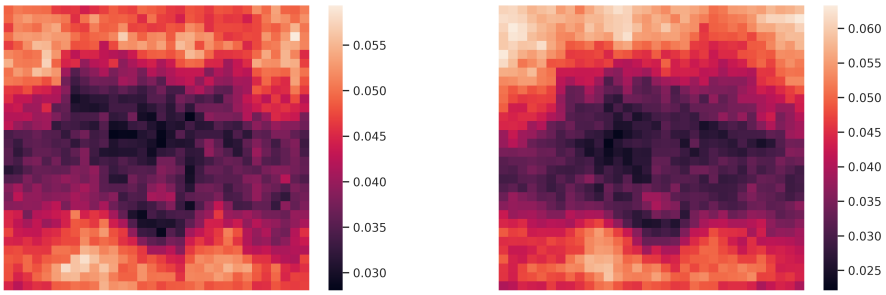


Figure 3.B.3: Aggregated heatmap of Xi-method explanations for images with true label deer. (left): single. (right): replicated

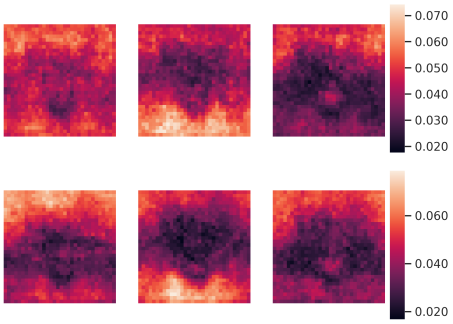


Figure 3.B.4: Channel heatmap of Xi-method explanations for images with true label deer. (above): single. (below): replicated

Baldassi, C., Borgs, C., Chayes, J. T., Ingrosso, A., Lucibello, C., Saglietti, L., and Zecchina, R. (2016). Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes. *Proceedings of the National Academy of Sciences*, 113(48):E7655–E7662.

Baldassi, C., Ingrosso, A., Lucibello, C., Saglietti, L., and Zecchina, R. (2015). Subdominant dense clusters allow for simple learning and high computational performance in neural networks with discrete synapses. *Physical review letters*, 115(12):128101.

Baucells, M. and Borgonovo, E. (2013). Invariant probabilistic sensitivity analysis. *Management Science*.

Binder, A., Bach, S., Montavon, G., Müller, K. R., and Samek, W. (2016). Layer-wise relevance propagation for deep neural network architectures. In *Lecture Notes in Electrical Engineering*, volume 376.

Boehmke, B., Greenwell, B., Boehmke, B., and Greenwell, B. (2020). Interpretable Machine Learning. *Hands-On Machine Learning with R*, pages 305–342.

Borgonovo, E. (2007). A new uncertainty importance measure. *Reliability Engineering and System Safety*, 92(6).

Borgonovo, E., Hazen, G. B., and Plischke, E. (2016). A Common Rationale for Global Sensitivity Measures and Their Estimation. *Risk Analysis*, 36(10):1871–1895.

Borgonovo, E. and Rabitti, G. (2020). From Tornado Diagrams to Effective Dimensions. *submitted*.

Borgonovo, E., Tarantola, S., Plischke, E., and Morris, M. D. (2014). Transformations and invariance in the sensitivity analysis of computer experiments. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*.

Borgonovo, E. and Ghidini, V. and Hahn, R. and Plischke, E. (2021). Dependence Measures for Classification: the Xi-Method. *Work in Progress*.

Caflisch, R., Morokoff, W., and Owen, A. (1997). Valuation of mortgage-backed securities using Brownian bridges to reduce effective dimension. *The Journal of Computational Finance*, 1(1).

Chaudhari, P., Baldassi, C., Zecchina, R., Soatto, S., Talwalkar, A., and Oberman, A. (2017). Parle: parallelizing stochastic gradient descent. *arXiv preprint arXiv:1707.00424*.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2019). Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018.

Efron, B. and Stein, C. (1981). The Jackknife Estimate of Variance. *The Annals of Statistics*, 9(3).

Erhan, D., Bengio, Y., Courville, A., and Vincent, P. (2009). Visualizing higher-layer features of a deep network. *Bernoulli*, (1341).

Hahn, R., Feinauer, C., and Borgonovo, E. (2021). The Mean Dimension of Neural Networks and what it reveals. *submitted*.

Hastie, T., Tibshirani, R., and Buja, A. (1994). Flexible discriminant analysis by optimal scoring. *Journal of the American Statistical Association*, 89(428):1255–1270.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13.

Hochreiter, S. and Schmidhuber, J. (1997). Flat minima. *Neural computation*, 9(1):1–42.

Hoyt, C. R. and Owen, A. B. (2020). Efficient estimation of the ANOVA mean dimension, with an application to neural net classification. *arXiv*, pages 1–26.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-January.

Iman, R. L. and Conover, W. J. (1987). A measure of top – down correlation. *Technometrics*, 29(3).

Iman, R. L. and Hora, S. C. (1990). A Robust Measure of Uncertainty Importance for Use in Fault Tree System Analysis. *Risk Analysis*, 10(3).

Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., and Bengio, S. (2019). Fantastic generalization measures and where to find them.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. *... Science Department, University of Toronto, Tech. ...*.

LeCun, Y. et al. (2015). Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14.

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. (2019). Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences of the United States of America*, 116(44).

Oakley, J. E. (2009). Decision-theoretic sensitivity analysis for complex computer models. *Technometrics*, 51(2).

Owen, A. B. (2003). The dimension distribution and quadrature test functions. *Statistica Sinica*.

Petsiuk, V., Das, A., and Saenko, K. (2019). RisE: Randomized input sampling for explanation of black-box models. In *British Machine Vision Conference 2018, BMVC 2018*.

Pittorino, F., Lucibello, C., Feinauer, C., Malatesta, E. M., Perugini, G., Baldassi, C., Negri, M., Demyanenko, E., and Zecchina, R. (2020). Entropic gradient descent algorithms and wide flat minima. *arXiv preprint arXiv:2006.07897*.

Plischke, E. and Borgonovo, E. (2020). Fighting the Curse of Sparsity: Probabilistic Sensitivity Measures From Cumulative Distribution Functions. *Risk Analysis*, 40(12):2639–2660.

Rabitz, H. and Aliş, Ö. F. (1999). General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25(2-3).

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 13-17-Augu, pages 1135–1144.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead.

Saltelli, A. (2002). Making best use of model evaluations to compute sensitivity indices. *Computer Physics Communications*, 145(2).

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-October.

Sobol', I. (1993). Sensitivity Estimates for Nonlinear Mathematical Models.

Zhang, S., Choromanska, A., and LeCun, Y. (2014). Deep learning with elastic averaging sgd. *arXiv preprint arXiv:1412.6651*.

Chapter 4

Neural Networks and Statistical Dependence: Does it Matter?

4.1 Introduction

Identifying the features that determine the response of a model is a fundamental ingredient for an interpretability analysis. It helps analysts with model simplification and dimensionality reduction, as well as to understand whether predictions are at risk of unfair discrimination (Dong and Rudin, 2020).

In a previous work by the authors, it was uncovered that probabilistic sensitivity measures may be useful in understanding the features that are important for artificial neural networks (ANNs) decisions (Borgonovo, E. and Ghidini, V. and Hahn, R. and Plischke, E., 2021). However, such work opens the question of whether statistical dependence is, per se, sufficient to explain an ANN behavior. In this chapter, we wish to explore the relationship between measures of statistical dependence and the behavior of artificial neural networks. Our goal is to investigate whether global sensitivity measures provide a sufficiently reliable indication about what a neural network considers important in a dataset. To do this, we investigate whether features that are considered statistically important at the dataset level

are also important for the prediction accuracy of convolutional neural networks (CNNs).

We perform a set of experiments to compare results of dataset importance measures with results of importance measures at the aggregate prediction level. We use two well-known image datasets, MNIST (LeCun et al., 1998) and Fashion-MNIST (Xiao et al., 2017). We test the effect of using alternative ways of pixel fixing and compare degradation plots obtained with measures of statistical dependence vs. aggregate Layerwise Relevance Propagation (LRP) vs. a deterministic structured deletion and vs. the selection of groups of pixels using principal component analysis (PCA). The remainder of the work is organized as follows. Section 4.2 presents the related literature. The elements and the results of the experiments are illustrated in Section 4.3 and 4.4. We conclude with a discussion in Section 4.5.

4.2 Related Literature

This section concisely reviews the two main research streams closer to our paper. Subsections 4.2.1 and 4.2.2 discuss respectively feature importance techniques and global statistical dependence. These research streams are vast and we do not claim exhaustiveness, and refer to the monographs of Boehmke et al. (2020) or reviews such as Dunson (2018), Begoli et al. (2019), Guidotti et al. (2018) and Rudin (2019) for broader overviews.

4.2.1 Feature Importance & Selection

Researchers have intensively investigated tools for the identification of important features in the context of machine learning predictions. Some of the recently developed methods are tailored to specific ML-models, some are ML-model independent. A notable representative of the first category is the split count feature importance measures of Breiman et al. (1984), tailored to regression trees. Representatives of the second category are methods such as Shapley values (Lundberg and Lee, 2017) or permutation importance (Breiman,

2001). However, permutation-based methods have come under intense scrutiny, recently. The investigations of Hooker and Mentch (2019); Apley and Zhu (2020); Molnar et al. (2020), show that, when features are strongly correlated, feature permutations can lead to unreliable predictions of the ML-model due to extrapolation problems. In the class of ML-method independent methods, we recall the important class of knockoffs (Barber and Candés, 2015) and Model-X knockoffs (Candès et al., 2018) methods. These are based on the intuition of constructing replicates of the original features for finding the smallest subset of the independent variables such that the response is conditionally independent of the remaining (other) independent variables. The most recent version allows application of the knockoff method to high-dimensional settings (with $p \geqslant n$; the number of parameters p is larger than the sample size n). An alternative technique is the Sure Independence Screening (SIS) (Fan and Lv, 2008), a feature selection procedure also aimed at identifying a subset of covariates according after the application of a linear model.

Murdoch et al. (2019) divide explanation techniques into families of individual prediction methods and families of dataset methods. To the first class belong methods such as Randomized Input Sampling for Explanation (RISE) (Petsiuk et al., 2018) and Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016). In RISE, the importance is evaluated according to the magnitude of the drop in accuracy after a single input image are perturbed by means of randomly generated binary masks. This allows the computation of a score for each pixel. The RISE technique is data aware, since it can only be applied to images, but is still model-agnostic. LIME employs a similar idea but can be applied to text and tabular data besides image data. In an image classification application, images are first divided into superpixels, and then the modified input is perturbed, obscuring one partition of the image at a time. An interpretable model (e.g. linear regression or tree) is learned using the binary vector of superpixels and the output of the original classifier. The analysis of the interpretable model reveals then the important patches in the image for the classifiers decision.

Another family of explanation methods are model aware techniques: they can only be applied to a specific model (in our case CNNs), since they exploit some characteristic of its structure (e.g. the gradient). Among them, we cite GradCAM (Selvaraju et al., 2017), Vanilla Gradient (Erhan et al., 2009), Guided Backpropagation (Springenberg et al., 2015) and Layerwise Relevance Propagation (LRP)(Bach et al., 2015; Binder et al., 2016). All these methods create heatmaps of importance scores for the pixels in an image.
In this work, we shall focus on LRP as representative of this class of methods. LRP is mainly applied to computer vision tasks but also to the analysis of text documents (Arras et al., 2016). As the name says, the idea is to propagate relevance backward through the network. Relevance values are the magnitude of the contribution of each pixel or intermediate neuron. The process starts at the activation values of the last layer and then back-propagates the relevance values, R, up to the input layer. It is designed to be a conservative technique, meaning that the magnitude of any output is conserved through the backpropagation process.

Further widely used approaches to measure feature importance in classification are based on "hiding" pixels and on evaluating the consequences on the prediction. However, there are alternative ways to hide a pixel: some authors use pixel removal, a procedure in which the pixel is taken out from the dataset (Hooker et al., 2019a). Others propose to assign a new color to the pixel, e.g., setting it to black or white or gray or random (color fixing, henceforth) (Petsiuk et al., 2018). In both cases (removal or color fixing) the CNN is retrained. However, in the complete removal case, we are retraining the CNN on a space of lower dimensionality, while in the fixing case we remain with the original dimensionality. To evaluate explainability techniques (Hooker et al., 2019b) propose the RemOve And Retrain (ROAR) method. ROAR essentially verifies the validity of the scores assigned to features by sequentially removing proportions of pixels (according to the feature importance measure of interest), generating synthetic data using this newly obtained distribution and then studying the resulting degradation of the accuracy of the (retrained) model. If a

method works well, it should rank first the most important features and their early removal should lead to a significant drop in accuracy in predictions. The ROAR procedure involves multiple retraining of the model on newly generated datasets at different degradation levels (i.e. with a different proportion of obscured pixels).

4.2.2 Probabilistic Sensitivity Measures

Probabilistic, or global, sensitivity measures are a class of statistical methods devised to measure the degree of statistical dependence between a target variable, say Y, and one or more covariates $(X_1, X_2, \ldots, X_n)$. In the literature, we find alternative definitions. Historically, we find regression-based approaches (Saltelli and Marivoet, 1990; Kleijnen and Helton, 1999), in which the standardized regression coefficients or correlation coefficients are used as measures of association. These approaches have been followed variance-based approaches in which the importance of X_i is measured in terms of the contribution to the variance of Y (Saltelli and Tarantola, 2002; Oakley and O'Hagan, 2004). There have been also recent investigations on distribution-based approaches, which consider importance without reference to any moment of the output distribution. These approaches involve indicators based on alternative measures of separation between the model output, among which the L^1-norm, the Kuiper, the Cramer von Mises distances and the family of Csiszar divergences (Da Veiga, 2015; Rahman, 2016).

Probabilistic sensitivity measures are part of the family of measures of statistical association. The problem of defining and measuring statistical dependence is central in statistics. For a thorough discussion of the conceptual aspects of the notion we refer to Wermuth and Cox (2014), as we cannot give a detailed account here due to space limitations. Nowadays, the appearance of big data has renewed interest in measures of statistical dependence, as they may help in reducing problem dimensionality as well as in increasing interpretability — see Pan et al. (2019), Pan et al. (2020), Shen et al. (2020), and Chatterjee (2020). While there is no best measure of statistical dependence, Renyi

(1959) and Mori and Szekely (2019) list a set of postulates aimed at pointing researchers towards some minimal properties that a measure of association should possess in order to be considered a solid measure of statistical dependence (see also the discussion in Chatterjee (2020)). Specifically, postulate D of Renyi (1959) suggests that a measure of statistical dependence should be null if and only if the random variable of interest, say Y_j, is independent of X_i. Among measures that satisfy this axiom, we find Chatterjee (2020) correlation coefficient, distance covariance (Szekely and Rizzo, 2017) and Hilbert Schmidt Independence Criterion (HSIC) (Da Veiga, 2015). A further class of measures of statistical dependence is based on a copula rationale (Plischke and Borgonovo, 2019). For further details on global sensitivity analysis and uncertainty quantification methods, we refer to works such as Saltelli et al. (2008) and Sullivan (2015). These techniques have been mostly studied in the computer experiment literature. Note that their definition, in principle, requires a double loop of Monte Carlo simulations, a design that cannot be applied to a single dataset. However, the one-sample (or given data) estimation technique allows one to compute these sensitivity measures also from available datasets. The key-intuition of a given data approach dates back to Pearson (1905), and has been applied to the calculation of variance-based sensitivity measures in Strong et al. (2012), of value of information (Strong and Oakley, 2013) and of distribution-based sensitivity measures (Plischke et al., 2013). We discuss this strategy in detail in Section 4.3.1.2.

This then has opened the use of global sensitivity measures also in machine learning and data science applications. Recently, Taverniers et al. (2020) propose the use of the Mutual Information for increasing interpretability in deep neural networks used in the context of multiscale systems. The idea is to rank features based on a probabilistic sensitivity measure that considers the entire realizations of the inputs. In Taverniers et al. (2020)'s work, a deep neural network is used to emulate the behavior of a complex system in a forecasting task. Taverniers et al. (2020)'s work offers a first direct bridge between probabilistic sensitivity measures and deep neural networks (DNNs).

4.3 Elements of the Experiments

We consider an image classification problem with a CNN as a classifier. The measures of statistical dependence are introduced in Section 4.3.1. In Section 4.3.2, we explain the neural network view and a concise description of the LRP method, the PCA approach, the structured deletion (StruDel) and random removal follows in Sections 4.3.3, 4.3.4, 4.3.5 and 4.3.6. The meaning of removing a feature is presented in Section 4.3.7.

4.3.1 Defining and Computing Probabilistic Sensitivity Measures for Supervised Classification

In Sections 4.3.1.1 and 4.3.1.2, we address the non-ordinal nature of the output and the availability of datasets large enough for their estimation, respectively.

4.3.1.1 Definition

Let $(\Omega, \mathcal{B}(\Omega), \mathbb{P})$ be a reference probability space and let $\mathcal{P}$ be the set of all probability measures on $(\Omega, \mathcal{B}(\Omega))$. Consider a function $\zeta : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ and any two probability measures $\mathbb{P}, \mathbb{Q} \in \mathcal{P}$. Following Glick (1975), we say that $\zeta(\mathbb{P}, \mathbb{Q})$ is a separation measure between $\mathbb{P}$ and $\mathbb{Q}$, if it satisfies the conditions that $\zeta(\mathbb{P}, \mathbb{Q}) \geqslant 0$ and $\zeta(\mathbb{P}, \mathbb{Q}) = 0$ if $\mathbb{P} = \mathbb{Q}$. Let also Y and X denote two random variables on $(\Omega, \mathcal{B}(\Omega), \mathbb{P})$, with cumulative distribution function F_Y, F_X. A probabilistic sensitivity measure of X with respect to Y can be defined as (Borgonovo et al., 2013):

Definition 1 (Probabilistic Sensitivity Measure)**.** *We call the quantity*

$$\xi_X^{\mathbf{Y}} = \mathbb{E}[\zeta(\mathbb{P}_Y, \mathbb{P}_{Y|X})] \tag{4.1}$$

the probabilistic sensitivity measure of X with respect to Y based on the separation measure $\zeta(\cdot, \cdot)$.

Equation (4.1) represents the average value of the separation between $\mathbb{P}_Y$ and $\mathbb{P}_{Y|X}$ across all possible realizations of X. Definition 1 establishes a common rationale that encompasses several probabilistic sensitivity measures. For instance, setting $\zeta(\mathbb{V}_Y, \mathbb{V}_{Y|X}) = \frac{(\mathbb{E}[Y]-\mathbb{E}[Y|X])^2}{\mathbb{V}[Y]}$ one obtains first order variance-based sensitivity measures that coincide with Pearson's correlation ratio. Or setting

$$\zeta^{\mathrm{KL}}(f_{Y|X}(y), f_{Y|X}(y)) = \int_{\mathbb{R}} f_{Y|X}(y) \log\left(\frac{f_{Y|X}(y)}{f_Y(y)}\right) dy,$$

one obtains as corresponding global sensitivity measure the mutual information of X, as in Taverniers et al. (2020).

The choice of the inner separator $\zeta(\cdot, \cdot)$ in Equation (4.1) determines the properties of the dependence measure. In particular, we have the following:

Suppose the analyst wishes to understand the degree of dependence of the target variable in a classification problem. The calculation of measures of statistical dependence requires some consideration in this case. In a supervised classification setting, the target is a set of labels with a set of discrete realizations as support. The elements of the support are not necessarily members of a numerical space and therefore an ordering of these elements may not be meaningful. This creates problems for the direct implementation of measures of statistical association whose definition require that the realizations of Y are in a partially ordered space.

We propose the following approach. Let L be the random variable denoting a label, let $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_{n_L}\}$ denote its support, with n_L number of labels. Let $\mathbf{p}_L$ be the probability mass function (pmf) of the labels and $\mathbf{p}_{L|X}$ the conditional pmf given that a feature or a feature group is fixed. (We recall that $\mathbf{p}_L$ is a probability mass function if $\mathbf{p} = [p_1, p_2, \ldots, p_{n_L}]$, $p_i \geqslant 0$ and $\sum_{i=1}^{n_L} p_i = 1$.) One can define the inner statistic that measures the effect of conditioning on the features using a measure of separation between the marginal and conditional pmfs. Let $\mathcal{P}^{\mathrm{pmf}}$ denote the set of all probability

mass functions on $(\mathcal{L}, \mathcal{B}(\mathcal{L}))$ and let $\zeta^{\text{pmf}}(\cdot,\cdot)$ denote a separation measurement between probability mass functions, $\zeta^{\text{pmf}} : \mathcal{P}^{\text{pmf}} \times \mathcal{P}^{\text{pmf}} \to \mathbb{R}$. Then, we can write

Definition 2. *We call a probabilistic sensitivity measure for classification, the quantity*

$$\xi_X^{Class} = \mathbb{E}_X[\zeta^{pmf}(\mathbf{p}_L, \mathbf{p}_{L|X})]. \tag{4.2}$$

Formally, Equation (4.2) is a particular case of Equation (4.1). However, in Equation (4.2), we explicitly consider the probability mass function (pmf) of the labels. This pmf is, in fact, defined without ambiguity when inputs are categorical. To illustrate, the probability that we extract an image with a house out of a set of 60,000 is the same, independently of whether the house is assigned label 1 or 10. However, the corresponding cumulative distribution function requires an additional convention, that is, we need to order the labels and then stick to such order. If an alternative order is used, we get an alternative cumulative distribution function. Using directly the pmf avoids the additional step of setting such a convention.

Alternative choices are available for the separation measurement $\zeta^{\text{pmf}}(\cdot,\cdot)$. We list a few in Table 1.

Notation	**Formula**	**Description**
$\zeta^1(\mathbf{p},\mathbf{q})$	$\sum_{l=1}^{n_L} \lvert p_l - q_l \rvert$	L1 distance
$\zeta^2(\mathbf{p},\mathbf{q})$	$\sum_{l=1}^{n_L} (p_l - q_l)^2$	L2 distance
$\zeta^{\text{KU}}(\mathbf{p},\mathbf{q})$	$\max_{l=1}^{n_L} (p_l - q_l) - \min_{l=1}^{n_L} (p_l - q_l)$	Kuiper distance
$\zeta^{\text{KL}}(\mathbf{p},\mathbf{q})$	$\sum_{l=1}^{n_L} q_l \log\left(\frac{q_l}{p_l}\right)$	Kullback-Leibler divergence
$\zeta^{\text{HL}}(\mathbf{p},\mathbf{q})$	$1 - \sum_{l=1}^{n_L} \sqrt{p_l q_l} = \frac{1}{2}\sum_{l=1}^{n_L} \left(\sqrt{p_l} - \sqrt{q_l}\right)^2$	Hellinger distance

Table 1: Different choices for separation measurement $\zeta^{\text{pmf}}(\mathbf{p},\mathbf{q})$. Here $\mathbf{p}$ and $\mathbf{q}$ are two pmf's of the same dimension, such that $\sum_{l=1}^{n_L} p_l = 1$ and $\sum_{l=1}^{n_L} q_l = 1$.

Note that the Hellinger and Kullback-Leibler separations above are special cases of the

power divergence with $\lambda = -\frac{1}{2}$ and $\lambda \to 0$, respectively.

Proposition 3. *Given $\mathcal{L}$, $\zeta^{pmf}(\cdot,\cdot)$ defined above, if $\zeta^{pmf}(\cdot,\cdot)$ is a separation measurement between mass functions, the following holds:*

1. $\xi_X^L \geqslant 0$
2. $\xi_X^L = 0$ *if and only if L is independent of X.*

Regarding the choice of the metric, we shall compare numerical experiments in the reminder to guide the choice. As can be expected, our numerical experiments show that alternative metrics produce quite similar ranking. Thus, while we shall focus on the metric denominated with Kuiper in Table 1, alternative separation measurements can be adopted by analysts.

4.3.1.2 Estimation

Let $Pr(\cdot)$ denote the probability of an event and L be the random variable denoting a random label. Let $\mathcal{X}_i$ denote the support of feature X_i, $i = 1, 2, ..., n_X$ and let also $\mathcal{K}_i = \{\mathcal{X}_i^1, \mathcal{X}_i^2, \ldots, \mathcal{X}_i^K\}$ denote a partition of $\mathcal{X}_i$, i.e., a finite or countable collection of subsets of $\mathcal{X}_i$ such that $\mathcal{X}_i = \cup_{k=1}^K \mathcal{X}_i^k$ and $\mathcal{X}_i^k \cap \mathcal{X}_i^j = \varnothing$, for $k \neq j = 1, 2, ..., K$. For the sensitivity measure in Equation (4.2), the following expression represents the equation of a given data estimator given partition $\mathcal{K}_i$:

$$\widehat{\xi}_i(\mathcal{K}_i) = \sum_{k=1}^{K} Pr(X_i \in \mathcal{X}_i^k)\zeta^{\text{pmf}}(\mathbf{p}_L, \mathbf{p}_{L|X_i \in \mathcal{X}_i^k}), \tag{4.3}$$

where $\mathbf{p}_{L|X_i \in \mathcal{X}_i^k} = [p^1_{L|X_i \in \mathcal{X}_i^k}, p^2_{L|X_i \in \mathcal{X}_i^k}, ..., p^{n_L}_{L|X_i \in \mathcal{X}_i^k}]$ denotes a conditional pmf where $p^r_{L|X_i \in \mathcal{X}_i^k} = Pr(L = l_r | X_i \in \mathcal{X}_i^k)$ for $i = 1, 2, ..., n_X$. In Equation (4.3), we have evidenced that the value of this estimator, $\widehat{\xi}_i(\mathcal{K}_i)$, depends on the choice of the partition. In the case X_i is a discrete random variable, the partition is immediately given by the discrete set of realizations of X_i. Thus, in this case $\widehat{\xi}_i(\mathcal{K}_i) = \xi_i$, provided that the marginal and conditional

distributions are known. In the case X_i is continuous, following the intuition of Pearson (1905), $\widehat{\xi}_i(\mathcal{K}_i)$ tends to ξ_{X_i} as $\mathcal{K}_i$ gets finer and finer. Equation (4.3) assumes the marginal and conditional pmfs are given. In practice, however, these pmf's need to be estimated from the data. Let N be the number of images in the dataset and N^r the number of those observations labeled with ℓ_r. Then, the entries of the empirical pmf $\widehat{\mathbf{p}}_L$ are defined by $\widehat{p}_L^r = \dfrac{N^r}{N}$, which is a consistent estimator of $p_L^r = Pr(L = \ell_r)$ by the law of large numbers. Let $\mathcal{K}_i^N = (\mathcal{X}_i^1, \mathcal{X}_i^2, \ldots, \mathcal{X}_i^{K(N)})$ be a finite partition of the input space and set $N_{X_i}^k$ to the number of input observations in $\mathcal{X}_i^k$ and $N_{X_i}^{r,k}$ to the corresponding number of observations with label equal to ℓ_r in $N_{X_i}^k$. Then, $\widehat{p}_{\mathcal{X}_i}^k = \dfrac{N_{X_i}^k}{N}$ is an estimator of $p_{X_i}^k = Pr(X_i \in \mathcal{X}_i^k)$, $k = 1, 2, ..., K(N)$, and

$$\widehat{p}_{L|X_i \in \mathcal{X}_i^k}^r = \frac{N_{X_i}^{r,k}}{N}$$

is an estimator of $p_{L|X_i \in \mathcal{X}_i^k}^r = Pr(L = \ell_r | X_i \in \mathcal{X}_i^k)$. All these estimators are consistent by the law of large numbers, and can be calculated from a given dataset. Then, the given data estimator becomes

$$\widehat{\xi}_i(\mathcal{K}_i) = \sum_{k=1}^{K} \widehat{p}_{\mathcal{X}_i}^k \zeta^{\text{pmf}}(\widehat{\mathbf{p}}_L, \widehat{\mathbf{p}}_{L|X_i \in \mathcal{X}_i^k}). \tag{4.4}$$

We say that the sequence $K(N)$ is associated with a proper refining strategy if the associated sequence of partition is a sequence of refining partitions and if $K(N)$ is such that $\lim_{N\to\infty} K(N) = \infty$ and $\lim_{N\to\infty} \frac{N}{K(N)} = 0$. Consider now the given data estimator

$$\widehat{\xi}_i(N) = \sum_{k=1}^{K(N)} \widehat{p}_{\mathcal{X}_i}^k \zeta^{\text{pmf}}(\widehat{\mathbf{p}}_L, \widehat{\mathbf{p}}_{L|X_i \in \mathcal{X}_i^k}). \tag{4.5}$$

4.3.2 Deep Neural Networks: A Probabilistic Sensitivity Viewpoint

In a supervised learning setup, we want to learn an input-output mapping such that

$$\hat{L} = g^*(\mathbf{X}, \mathbf{w}), \tag{4.6}$$

where g^* denotes the model, that returns a predicted label $\hat{L}$, and $\mathbf{w}$ a vector of parameters. In our case, g^* represents a neural network in particular.

Deep neural networks are the common model of choice for many complex tasks such as classification; see Goodfellow et al. (2016) for an overview of deep neural networks.

From a general viewpoint, an artificial neural network for image classification can be represented as a mapping similar to the type in Equation (4.6) with $g : \mathbb{R}^{n_X} \times \mathbb{R}^{n_W} \to \mathbb{R}^{n_L}$, where n_X is the number of features, n_W is the number of parameters, consisting of weights and biases, and n_L is the number of targets. Thus, the model g outputs a score or a probability for each class of labels and g^* simply returns the most likely label as prediction.

In an image classification problem, the number of features n_X is equal to the number of pixels in each image. Say that the image has a resolution $n_V \times n_H$, where n_V and n_H are the numbers of vertical and horizontal pixels, respectively. Then, in the case of a color image, we have that $n_X = 3 \times n_V \times n_H$, where the first multiplier comes from the 3 RBG channel.

The form of the mapping g is then learned from the data. The determination of the input-output mapping is part of an optimization problem (Gambella et al., 2020).

The parameters $\mathbf{w} = \{w_1, w_2, ..., w_{n_W}\}$ in supervised learning are the solution of a data-driven optimization problem of the form:

$$\min_{\mathbf{w}} \mathbb{E}[\mathcal{C}(L, g(\mathbf{X}, \mathbf{w}))], \tag{4.7}$$

where $\mathcal{C}(L, g(\mathbf{X}, \mathbf{w}))$ is a suitably defined loss function, with $\mathcal{C} : \mathbb{R} \times \mathbb{R}^{n_L} \to \mathbb{R}$. Let us denote with $\mathbf{w}^*$ the parameters that solve Problem 4.7. Please see Gambella et al. (2020), Higham and Higham (2019) for ample discussions on techniques to solve Problem 4.7. Once $\mathbf{w}$ is fixed at $\mathbf{w}^*$, we are dealing with the input-output mapping

$$Y = g(\mathbf{X}, \mathbf{w}^*). \tag{4.8}$$

That is, in training a neural network, once we set structural elements such as the number of neurons and layers and the activation functions, it is only after we assign specific values to the weights that we determine the input-output mapping uniquely.

Coming to the problem of this chapter, we would argue that if a label L shows a strong dependence on a feature X_i in the data, then also the prediction $\hat{Y}$ of a well-trained neural network model g^* might evidence such dependence X_i. Thus, we wish to answer the question of whether a feature which is statistically important for an input-output mapping is also important for the machine learning model. Unfortunately, as it often happens in machine learning, this assertion cannot be proven analytically or universally. In particular, because the true underlying relationship is unknown, it is impossible to prove that features which are 'important' in a dataset are also 'important' for a network. However, the founding principle of using any machine learning model is that the model (the network in this case) well approximates the true input-output mapping (if it did not, why should it be used?). If this is true, then what is statistically important in the data should not be so different from what is important for the net. We formulate a way to verify/falsify this intuition in the reminder of the work.

4.3.3 Layerwise Relevance Propagation

LRP aims at finding the most important pixels of an image for a neural network prediction (Bach et al., 2015). The methodology is based on the 'graph-like' architecture of CNN's.

The function for neuron j in layer $l+1$ is of the form

$$a_j^{l+1} = \sigma_{l+1}(\sum_i a_i^l w_{ij}^{l+1} + b_j^{l+1}), \tag{4.9}$$

where σ_{l+1} is the activation function and the sum runs over all lower layer neurons connected to neuron j. As the name suggests, the very same graph structure is used to redistribute relevance of the model output backward through the net. The so-called pixel-wise relevance scores can be found using the local redistribution rule:

$$R_i^l = \sum_j \frac{z_{ij}}{\sum_{i'} z_{i'j}} R_j^{l+1} \tag{4.10}$$

with $z_{ij} = x_i^l w_{ij}^{l+1}$, where the sum in j runs over all upper-layer neurons to which neuron i contributes. This rule is then applied in a step-wise fashion starting from the output and ending in the input layer, giving a relevance score to each pixel.

The application of this rule in a backward pass produces a relevance map that satisfies the desired conservation property $\sum_p R_p^1 = g(x)$, where $g(x)$ is the model output. Hence, the sum of the relevance scores of all pixels will be equal to the relevance score of the model output.

The LRP provides a relevance score at the individual image level. Because we are interested in 'global' properties of the neural networks, we aggregate the relevance score per pixel over all images, by taking the mean. The dataset LRP feature importance of a pixel is then given by its aggregated score.

4.3.4 Principal Component Analysis

Principal Components Analysis (PCA) (Hastie et al., 2001; James et al., 2013) is a dimensionality reduction technique aimed at finding the best q components (where $q \leqslant d$, with d as original dimensionality of the input) which are orthogonal and that maximally capture the variability in the original data. Consider an $n \times d$ data matrix, X, with

column-wise zero empirical mean. The method transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. To do so, PCA solves the following optimization problem to find the first component $\phi_1 = (\phi_{11}, \ldots, \phi_{d1})$, which is constrained to be a unit vector:

$$argmax_{\phi_{11},\ldots,\phi_{d1}} \{\frac{1}{n}\sum_{i=1}^{n}(\phi_1 x_{(i)})^2\}, \tag{4.11}$$

where we n is the sample size, d the dimension of the feature space and $x_{(i)}$ is the i-th row vector of X. Further components are found by subtracting the previous principal components from the data and computing the weight vector which extracts the maximum variance from this new data matrix. Hence the k-th component can be found by computing the new data matrix

$$X^{(k)} = X - \sum_{s=1}^{k-1} X\phi_s\phi_s^T \tag{4.12}$$

and solving the optimization problem

$$argmax_{\phi_k} \{\frac{1}{n}\sum_{i=1}^{n}(\phi_1 x_{(i)}^{(k)})^2\}. \tag{4.13}$$

We exploit this popular method to gain another benchmark for feature importance. Note that this importance is found by considering only the input, ignoring the output. The 'feature importance' methodology based on PCA takes the most important component, which is the one that explains the highest ratio of the variability in the original data, and extracts the 50 original feature that influence this component the most. The process repeats similarly with the second most important component and then with the third most important component and so on. For each component extract the 50 most influential original features that were not yet previously extracted. This approach yields a ranking of

original features according to the importance based on their order of appearance in the PCA components.

4.3.5 Structured Deletion

This removal follows a simple deterministic path: Pixels are removed in a spiral pattern starting from the very center of an image. The rationale of using this ad-hoc method is based on the fact that we are dealing with centered images. We may imagine that, if images are centered and portray digits, pixels at the center of the image may be more informative that pixels in the outside parts. We refer to this approach as structured deletetion (StruDel).

4.3.6 Random Removal

The random removal approach serves as a benchmark to compare the other approaches. A pixel is drawn uniformly from all pixels of the image not drawn yet. This pixel is then removed and the next pixel is drawn.

4.3.7 Feature Removal and Degradation Plots

This subsection describes how we 'remove' a feature and how we evaluate the importance.

A degradation plot visualizes the evolution of the model accuracy with increasing number of masked/removed pixels starting with the most important ones. The vertical axis shows the accuracy and the horizontal one the number of removed pixels. The steeper the drop in accuracy, the better the feature importance measure in finding relevant pixels for the network's decision.

We use the degradation plots to analyze the question of whether neural networks 'see' statistical dependence by comparing degradation curves corresponding to different importance measures. Degradation plots are often used to compare the goodness of feature

importance methods (such as LRP, integrated Gradient, etc.). They help in evaluating the robustness and stability of black-box models by revealing how much of the neural network's decision depends on only few inputs.

We compare rankings induced by the probabilistic sensitivity measures introduced in Section 4.3.1 that are based solely on the input-output mapping with the rankings induced by the dataset LRP feature importance measures from Section 4.3.3. As benchmarks and for additional comparison we add the degradation curves of the PCA approach, the StruDel approach and the random pixel removal.

Now, when we consider pixel removal, we would change the input space if the pixel is actually completely removed. Necessarily we would deal with a different input-output mapping as discussed before. The complete removal would require a retraining of the model using a lower input dimensionality and by that we could not easily compare different 'feature importance' methods as needed in the next sections. To preserve dimensionality, it is suggested that the pixel is not actually taken out but it is assigned an alternative color. In this respect, we note that there is not a univocal way to choose the assignment and four different approaches of 'removing' a pixel. Thus, rather than really removing a pixel, we change its color to black, white, grey or a random greyscale. The idea is to mask the information coming from this pixel. Masking a pixel allows one to use the same trained model without changing the dimensionality (Petsiuk et al., 2018).

4.4 Numerical Experiments

In order to evaluate whether features that are statistically important are also important for the decision made by the neural network, we perform a series of experiments on the MNIST (LeCun et al., 1998) and FASHION-MNIST (Xiao et al., 2017) datasets. The well-known MNIST dataset is a database of handwritten digits with a training set of 60,000 examples, and a test set of 10,000 examples. The FASHION-MNIST dataset from

Zalando Research consists as well of a training set of 60,000 examples and a test set of 10,000 examples. It is thought of as a direct drop-in replacement for the original MNIST dataset. So images in both databases are 28×28 grayscale images, associated with a label from 10 classes. The MNIST images have simply the digits from 0 to 9 as labels, while the FASHION-MNIST labels are products from the fashion industry such as pullover, trouser, bags, etc.

The classification models are trained using a convolutional neural network, LeNet LeCun et al. (1998). The LeNet architecture is a 7 layer neural network consisting of the input layer, 2 convolutional layers each followed by a pooling layer and another convolutional layer followed by the output layer.

We proceed by performing experiments only with probabilistic sensitivity measures in Section 4.4.1, and we add LRP, STRUDEL and RANDOM for the further comparison using degradation plots in Section 4.4.2.

4.4.1 Comparison of the Ranking Using Alternative Distances

We start with analyzing feature importance measures coming from global sensitivity analysis. The importance measures are introduced in Section 4.3.1. This first evaluation aims at finding similarities or differences among these measures. We compare the methods of feature importance using two correlation indices, namely the well-known Spearman's rank and the Savage scores correlation coefficients (Iman and Conover, 1987).

The Savage score of feature i is defined as $s_i = \sum_{j=r_i}^{d} \frac{1}{j}$, where d is the total number of features and r_i is the rank of feature i. Hence, the Savage score correlation coefficient does not compute the correlation between the ordinary ranks but places more weight on highly ranked features. The intuition is that the Savage scores might provide a better basis for quantifying agreement, when the analyst wishes to place emphasis on the most important rather than on the least relevant features. In an image classification task, the exact order of low ranked pixels should not be the focus of the analysis since they might have assigned

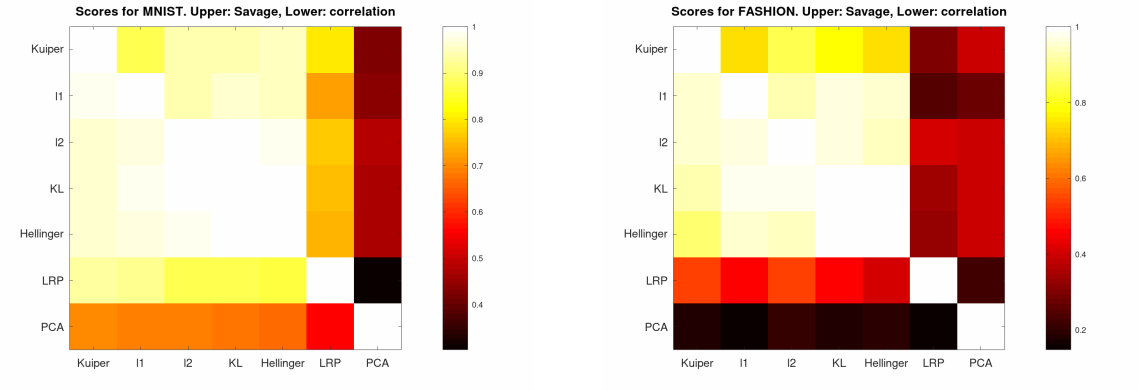
a very similar (low) value of importance, while the order of the most important pixels is crucial for understanding a network's decision.

For each sensitivity measure we compare the corresponding ranks. Figure 1a presents results for the MNIST database. The correlation coefficients are visualized using heatmaps. The little squares below the diagonal line represent the Spearman's rank correlation coefficients, while the ones above the diagonal line are representing the Savage score correlation coefficients. For example, the color of the square in the first column (L1-distance based probabilistic sensitivity measure) and fourth row (Kullback-Leibler) indicates a Spearman's rank correlation coefficient around 0.85.

We observe that all probabilistic importance measures produce a quite similar ranking of the features as the high correlation coefficients show. They are consistently greater than 0.8 for both correlation scores. For comparison, we have also included the newly defined PCA ranking criterion and the ranking according to LRP. The correlation between rankings induced by PCA importance measure and probabilistic sensitivity measures is significantly lower (around 0.6-0.7 for Spearman's rank correlation and 0.3-0.5 for Savage scores correlation). The comparison including LRP and PCA among others is discussed in greater detail in Section 4.4.2.

Comparing the rankings induced by the probabilistic sensitivity measures, the large values of Spearman's rank correlation coefficients indicate a high agreement of the rankings over all pixels and the high Savage score correlation coefficients assure a high agreement of pixels that have been assigned a high feature importance score by the probabilistic sensitivity measures.

Figure 1b displays the heatmap of the two correlation coefficients for the FashionMNIST database. The Spearman's rank correlation coefficients indicate a lower agreement on the rankings induced by probabilistic sensitivity measures compared to MNIST, but still the correlation coefficients are consistently larger than 0.8. The Savage score correlation coefficients show a lower agreement compared to MNIST as well, but still with values



(a) MNIST Database (b) FashionMNIST Database

Figure 1: Heatmaps of Spearman's rank correlation coefficients (triangle below the main diagonal) and Savage score correlation coefficients (triangle above the main diagonal) of the presented 'feature importance measures' using the MNIST Database and FashionMNIST database. GSA feature importance measures are based on Kuiper, L1, L2 and Hellinger distance and Kullback-Leibler divergence. For further comparison, the aggregated LRP and PCA are added to the analysis.

above 0.6. Hence, probabilistic sensitivity measures lead to similar ranking regarding the most important pixels. For this reason, we decide to use the Kuiper-based sensitivity measure for the further analysis as a representative for the class of probabilistic sensitivity measures. We call it Kuiper importance measure, henceforth. Indeed we have checked that the results do not change much using any of the other global sensitivity measures.

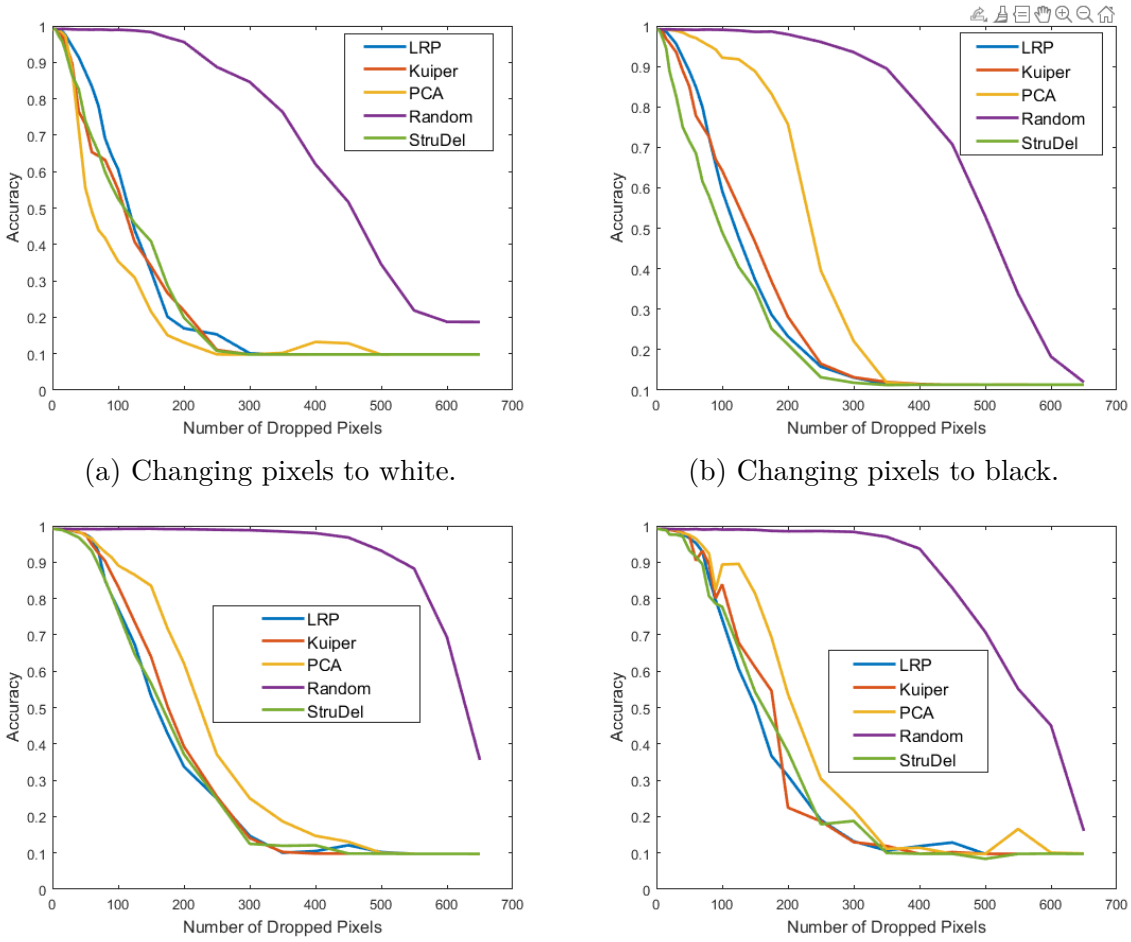
4.4.2 LRP vs. StruDel vs. GSA

In order to evaluate if inputs that have high statistically dependence with the output are also important for the neural network's decision, we start a comparison based on degradation plots.

As a method to represent which pixels are important for the network, we choose the well-known LRP in an aggregated form as described in Section 4.3.3. The rankings induced by importance measures using LRP are compared to the ones induced by the Kuiper-based probabilistic sensitivity measures. The Kuiper importance measure represents the statistical dependence in the data and requires only the images and the corresponding

labels; no model is required to compute it. For further comparison, we add degradation curves based on the rankings induced by the PCA approach (Section 4.3.4), the StruDel approach (Section 4.3.5) and the random removal approach (Section 4.3.6).

Let us start with the analysis using the MNIST database. As we said in the previous section, there are different ways pixels can be 'removed'. We start with the case where pixels are changed to white. Figure 2a shows the corresponding degradation plot for rankings induced by the Kuiper, LRP, PCA, StruDel and random removal importance measures. The abscissa shows how many pixels were changed and the ordinate gives the accuracy in prediction after the pixels have been 'removed'. The most important pixel is changed first, followed by the others in accordance with the rank induced by the corresponding importance measure. After the first 50 'removed' pixels, we register a similar drop of about 20% in accuracy for all methods with the exception of the aggregated LRP, which drops only about 10%. As no surprise, the random removal is far off from the other methods consistently for all plots. Randomly removing pixels does decrease the accuracy very little until up to 200 'removed' pixels. After the first 50 'removed' pixels, PCA seems to clearly outperform the other importance measures. Again, outperforming here means that removing pixels according to our PCA criterion leads to a higher drop in accuracy than removing them following the ranking induced by the other methods. After 'removing' 100 pixels, we loose around 65% in accuracy with the PCA method and 30-40% with Kuiper, LRP and StruDel importance measures. The good performance of StruDel through all of our experiments is no surprise since the images in MNIST are centered.



(a) Changing pixels to white.

(b) Changing pixels to black.

(c) Changing pixels to grey.

(d) Changing pixels to random.

Figure 2: Degradation plots for MNIST dataset according to rankings induced by Kuiper, aggregated LRP (LRP), PCA, StruDel and random removal 'importance measures'. Pixels are changed to (a) white, (b) black, (c) grey and (d) random greyscale.

Figure 2b shows the results when pixels are changed to black. In this case, PCA performs worse than all other methods, with the exclusion of random removal, signaling almost no drop at all in accuracy. StruDel performs best which is reasonable because images are centered and turning pixels black in a spiral way starting from the center might quickly deceive the network. After 50 'removed' pixels, we register a decrease of about 20% in accuracy using the StruDel approach. The degradation curves of the LRP and the Kuiper importance measure produce similar results in this case leading to an accuracy drop of 10%.

In general, changing pixels to black leads to a far smaller drop in accuracy compared to the change to white after 100 'removed' pixels. This indicates that the majority of the most important pixels are black for all employed methods. This result is in accordance with intuition because the images consist of black numbers on a white background.

Changing pixels to grey (Figure 2c) leads to similar results as changing them to black, but with a much smaller magnitude of accuracy drop and a smaller difference between the used methods. Indeed, the degradation curves according to the LRP, Kuiper and StruDel importance measures follow a very similar course. After 50 'removed' pixels, all methods lead to a very little decrease in accuracy of only about a few percentage points, while after 100 pixels, the 'removed' pixels according to the rankings induced by LRP, Kuiper and StruDel importance measures lead to a drop in accuracy of about 20% and the ranking induced by the PCA method leads to a decrease of about 10% in accuracy.

Changing pixels to a random value gives a slightly different picture. As expected the accuracy still drops with higher number of changed pixels but the curves are less smooth compared to the previous removal techniques. Moreover, we observe some unexpected behavior with the PCA method. After having changed 100 pixels, there is a strong increase in accuracy instead of the expected decrease. Again the degradation curves according to the LRP, Kuiper and StruDel importance measures follow a similar course.

Overall, global sensitivity measures seem to perform well in finding important pixels for the network decision over the experiments with MNIST, although it is clear that other aspects besides statistical dependence are important for the network decisions.

We now present results of similar experiments performed on the FASHION-MNIST database (Figures 3a-3d). We register two main differences for all four pixel removing techniques; a greater difference among the used degradation curves and better performance of the degradation curve induced by the random removal compared to the experiments with MNIST.

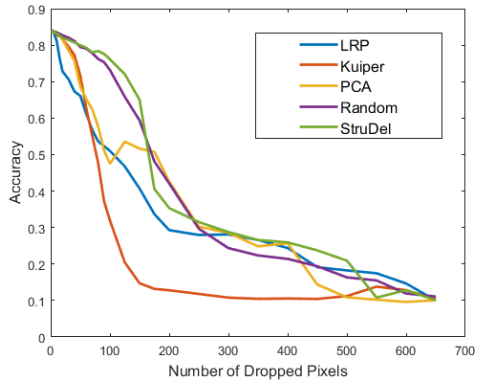
When changing pixels to white (Figure 3a), the StruDel method works surprisingly

even worse than the random removal method in terms of induced drop in accuracy. The LRP method induces the biggest drop in accuracy up to 80 changed pixels, but again the degradation plot induced by the Kuiper importance measures follows very close the one induced the LRP importance measures. After 80 'removed' pixels, the loss in accuracy of the LRP, PCA and Kuiper methods is about 20% and with further 'removed' pixels the degradation curve induced by the Kuiper importance measure shows by far the steepest drop.

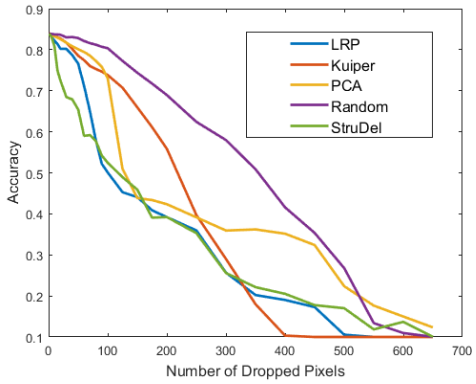
The results differ when changing pixels to black (Figure 3b). StruDel is outperforming other methods when only a small number of pixels is 'blackened'. After 100 'removed' pixels, LRP induces a drop in accuracy of about 35%, while the Kuiper importance measure leads only to a reduction of about 10%.

Figure 3c displays the results when pixels are changed to grey. The degradation curve induced by the LRP method leads to the strongest drop in accuracy when 'removing' up to 170 pixels. The degradation curve induced by the Kuiper importance measure comes closer and after 'removing' ca. 170 pixels, the two curves intersect and the latter one shows a stronger drop in accuracy for further 'removed' pixels. We register a drop in accuracy after 100 'removed' pixels of about 25% for the LRP method and between 5% and 8% for the Kuiper-based, PCA and StruDel method.

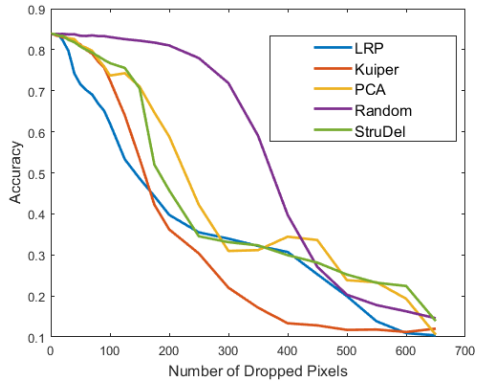
We register similar results from the random color change experiment (Figure 3d). Again the LRP method leads to the strongest decrease in accuracy when 'removing' up to 120 pixels according to the induced ranking. After 120 pixels, the degradation curve induced by the Kuiper importance measure intersects with the one induced by the LRP method. After 100 'removed' pixels, the LRP, Kuiper, PCA, StruDel and random removal importance measure lead to a drop in accuracy of about 25%, 15%, 10%, 5% and 2%.



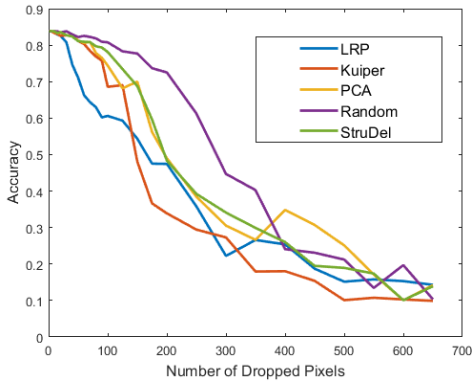
(a) Changing pixels to white.



(b) Changing pixels to black.



(c) Changing pixels to grey.



(d) Changing pixels to random greyscale.

Figure 3: Degradation plots for FASHION-MNIST dataset according to rankings induced by Kuiper, aggregated LRP (LRP), PCA, StruDel and random removal 'importance measures'. Pixels are changed to (a) white, (b) black, (c) grey and (d) random greyscale.

4.5 Conclusions

We have investigated the question of whether neural networks 'see' statistical dependence. For this aim, we have compared the drop in accuracy of a trained model when 'removing' pixels according to the rankings induced by different importance measures.

As a representative for feature importance of a neural network, we choose an aggregated LRP approach, since LRP is specifically designed to explain neural network's decision in an image classification task. Further 'importance measures' were added as a benchmark and for a broader comparison. We highlight the fact that our work is not aimed at feature

selection: we want to study whether features that are strongly statistically correlated with the target retain their relevance also for a black-box model.

There is no straightforward answer to give. The systematic analysis shows that results vary according to the chosen pixel removal method and also depend on the dataset. Although the rankings induced by the LRP and the Kuiper feature importance measures do not coincide, it emerges that features having a strong statistical dependence with the output are also important for the trained neural network. Overall the probabilistic sensitivity measures and the importance scores of LRP lead in most cases to similar drops in accuracy indicating a comparable success in finding the most important pixels for a network's decision.

A future research stream can apply our approach to the investigation of the so-called artificial intelligence brittleness (AI brittleness, henceforth). AI brittleness refers to dataset-independence (Seewald, 2012), i.e., the ability of a network trained on a dataset to perform well on a similar, but different dataset. To fix ideas, Seewald (2012) registers AI brittleness when considering alternative AI tools for classification in the MNIST and USPS (Hull, 1994) for the recognition of handwritten digits (e.g., a classifier well-trained on MNIST does not perform as well on USPS). Our results aid in the interpretation of such findings. These datasets differ substantially in terms of statistical dependence. Thus, pixels that are important in MNIST or USPS may not be as important in the European digit dataset for a neural network decision.

References

Apley, D. W. and Zhu, J. (2020). Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 82(4):1059–1086.

Arras, L., Horn, F., Montavon, G., Müller, K.-R., and Samek, W. (2016). What is Relevant

in a Text Document? *PLoS One*, 12(8):1–19.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K. R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7):1–46.

Barber, R. F. and Candés, E. J. (2015). Controlling the false discovery rate via knockoffs. *Annals of Statistics*, 43(5):2055–2085.

Begoli, E., Bhattacharya, T., and Kusnezov, D. (2019). The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence*, 1(1):20–23.

Binder, A., Bach, S., Montavon, G., Müller, K. R., and Samek, W. (2016). Layer-wise relevance propagation for deep neural network architectures. *Lecture Notes in Electrical Engineering*, 376:913–922.

Boehmke, B., Greenwell, B., Boehmke, B., and Greenwell, B. (2020). Interpretable Machine Learning. *Hands-On Machine Learning with R*, pages 305–342.

Borgonovo, E., Hazen, G., and Plischke, E. (2013). A Common Rationale for Global Sensitivity Analysis. In Steenbergen, R. D. M. ., Van Gelder, P., Miraglia, S., and Vrouwenvelder, A. C. W. M. T., editors, *Proceedings of the 2013 ESREL Conference, Amsterdam*, pages 3255–3260.

Borgonovo, E. and Ghidini, V. and Hahn, R. and Plischke, E. (2021). Dependence Measures for Classification: the Xi-Method. *Work in Progress*.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and C.J., S. (1984). *Classification and regression trees*. Chapman&Hall, New York.

Candès, E., Fan, Y., Janson, L., and Lv, J. (2018). Panning for gold: 'model-X' knockoffs for high dimensional controlled variable selection. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 80(3):551–577.

Chatterjee, S. (2020). A New Coefficient of Correlation. *Journal of the American Statistical Association*.

Da Veiga, S. (2015). Global Sensitivity Analysis with Dependence Measures. *Journal of Statistical Computation and Simulation*, 85(7):1283–1305.

Dong, J. and Rudin, C. (2020). Exploring the cloud of variable importance for the set of all good models. *Nature Machine Intelligence*, 2(12):810–824.

Dunson, D. B. (2018). Statistics in the Big Data Era: Failures of the Machine. *Statistics and Probability Letters*, 136:4–9.

Erhan, D., Bengio, Y., Courville, A., and Vincent, P. (2009). Visualizing higher-layer features of a deep network. *Bernoulli*, (1341):1–13.

Fan, J. and Lv, J. (2008). Sure Independence Screening for Ultrahigh Dimensional Feature Space Published by : Wiley for the Royal Statistical Society Stable URL : http://www.jstor.org/stable/20203862 Linked references are available on JSTOR for this article : Sure independence scre. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 70(5):849–911.

Gambella, C., Ghaddar, B., and Naoum-Sawaya, J. (2020). Optimization Problems for Machine Learning: A Survey. *European Journal of Operational Research*, Forthcomin:1–83.

Glick, N. (1975). Measurements of separation among probability densities or random variables. *Canadian Journal of Statistics*, 3(2):267–276.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018). A Survey of Methods for Explaining Black Box Models. *ACM Computing Surveys*, 51(5):1–42.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Higham, C. F. and Higham, D. J. (2019). Deep Learning: An Introduction for Applied Mathematicians. *Siam Review*, 61(4):860–891.

Hooker, G. and Mentch, L. (2019). Please Stop Permuting Features: An Explanation and Alternatives. *ArXiv*, 1905(03151):1–15.

Hooker, S., Erhan, D., Kindermans, P.-J., and Kim, B. (2019a). A Benchmark for Interpretability Methods in Deep Neural Networks. *ArXiv*, 1806(10758v3):1–17.

Hooker, S., Erhan, D., Kindermans, P. J., and Kim, B. (2019b). A benchmark for interpretability methods in deep neural networks. *Advances in Neural Information Processing Systems*, 32(NeurIPS).

Hull, J. J. (1994). A Database for Handwritten Text Recognition Research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5).

Iman, R. and Conover, W. (1987). A measure of top-down correlation. *Technometrics*, 29(3):351–357.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

Kleijnen, J. and Helton, J. (1999). Statistical analyses of scatterplots to identify important factors in large-scale simulations, 2: robustness of techniques. *Reliability Engineering & System Safety*, 65(2):187–197.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 4766–4775.

Molnar, C., Koenig, G., Bischl, B., and Casalicchio, G. (2020). Model-agnostic Feature Importance and Effects with Dependent Features: A Conditional Subgroup Approach. *ArXiv*, :2006.0462(v1):1–20.

Mori, T. and Szekely, G. (2019). Four Simple Axioms of Dependence Measures. *Metrika*, 82:1–16.

Murdoch, W. J., Signh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. (2019). Definitions, Methods and Applications in interpretabile Machine Learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080.

Oakley, J. and O'Hagan, A. (2004). Probabilistic Sensitivity Analysis of Complex Models: a Bayesian Approach. *Journal of the Royal Statistical Society, Series B*, 66(3):751–769.

Pan, W., Wang, X., Xiao, W., and Zhu, H. (2019). A Generic Sure Independence Screening Procedure. *Journal of the American Statistical Association*, 114(526):928–937.

Pan, W., Wang, X., Zhang, H., Zhu, H., and Zhu, J. (2020). Ball Covariance: A Generic Measure of Dependence in Banach Space. *Journal of the American Statistical Association*, 115(529):307–317.

Pearson, K. (1905). *On the General Theory of Skew Correlation and Non-linear Regression*, volume XIV of *Mathematical Contributions to the Theory of Evolution, Drapers' Company Research Memoirs*. Dulau & Co., London.

Petsiuk, V., Das, A., and Saenko, K. (2018). RISE: Randomized input sampling for explanation of black-box models. *arXiv*, 1.

Plischke, E. and Borgonovo, E. (2019). Copula theory and probabilistic sensitivity analysis: Is there a connection? *European Journal of Operational Research*, 277(3):1046–1059.

Plischke, E., Borgonovo, E., and Smith, C. (2013). Global Sensitivity Measures from Given Data. *European Journal of Operational Research*, 226(3):536–550.

Rahman, S. (2016). The f-Sensitivity Index. *SIAM/ASA Journal on Uncertainty Quantification*, 4(1):130–162.

Renyi, A. (1959). On Measures of Statistical Dependence. *Acta Mathematica Academiae Scientiarum Hungarica*, 10:441–451.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why should i trust you?" Explaining the predictions of any classifier. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-Augu:1135–1144.

Rudin, C. (2019). Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 1(5):206–215.

Saltelli, A. and Marivoet, J. (1990). Non-parametric statistics in sensitivity analysis for model output: A comparison of selected techniques. *Reliability Engineering & System Safety*, 28(2):229–253.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. (2008). *Global Sensitivity Analysis – The Primer*. Chichester.

Saltelli, A. and Tarantola, S. (2002). On the Relative Importance of Input Factors in Mathematical Models: Safety Assessment for Nuclear Waste Disposal. *Journal of the American Statistical Association*, 97(459):702–709.

Seewald, A. K. (2012). On the Brittleness of Handwritten Digit RecognitionModels. *ISRN Machine Vision*, pages 1–10.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision*, 128(2):336–359.

Shen, C., Priebe, C. E., and Vogelstein, J. T. (2020). From Distance Correlation to Multiscale Graph Correlation. *Journal of the American Statistical Association*, 115(529):280–291.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, pages 1–14.

Strong, M. and Oakley, J. (2013). An efficient method for computing partial expected value of perfect information for correlated inputs. *Medical Decision-Making*, 33:755–766.

Strong, M., Oakley, J. E., and Chilcott, J. (2012). Managing Structural Uncertainty in Health Economic Decision Models: a Discrepancy Approach. *Journal of the Royal Statistical Society, Series C*, 61(1):25–45.

Sullivan, T. (2015). *Introduction to Uncertainty Quantification*. Springer Verlag.

Szekely, G. and Rizzo, M. (2017). The Energy of Data. *Annual Reviews*, 4:447–479.

Taverniers, S., Hallb, E. J., Katsoulakisc, M. A., and Tartakovsky, D. M. (2020). Mutual Information for Explainable Deep Learning of Multiscale Systems. *ArXiv*, 2009.04570(v1):1–22.

Wermuth, N. and Cox, D. R. (2014). Statistical Dependence and Independence. In *Wiley StatsRef: Statistics Reference Online*. American Cancer Society.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Chapter 5

Appendix

5.1 Code: Estimating Mean Dimension of Neural Networks

The code to train a model is printed in Section 5.1.1. Section 5.1.2 displays the code to estimate the mean dimension given a trained model.

5.1.1 Training a Neural Network

```
import torch
import argparse
import torch.optim as optim
from torch.utils.data import SubsetRandomSampler, DataLoader
import time
import os
from tqdm import tqdm
import pandas as pd
import numpy as np
```

```
import tempfile
from shutil import copyfile

from sacred import Experiment
from sacred.commands import print_config
ex = Experiment('RobustDNN')
from sacred import SETTINGS
SETTINGS['CAPTURE_MODE'] = 'no'
import pdb

# add script's parent folder to path
import os, sys
current_dir = os.path.basename(os.path.dirname(os.path.realpath(
    __file__)))
sys.path.insert(0, os.path.dirname(current_dir))

# SacredDNN imports
from sacreddnn.models.robust import RobustNet, RobustDataLoader
from sacreddnn.parse_args import get_dataset, get_loss_function,
     get_model_type, get_optimizer
from sacreddnn.utils import num_params, l2_norm,
    run_and_config_to_path,\
                file_observer_dir, to_gpuid_string,
                   take_n_per_class

def train(loss, model, device, train_loader, optimizer):
     model.train()
```

```
    t = tqdm(train_loader) # progress bar integration
    train_loss, accuracy, ndata = 0, 0, 0
    for data, target in t:
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        l = loss(output, target)
        l += model.coupling_loss()
        l.backward()
        optimizer.step()

        train_loss += l.item()*len(data)
        pred = output.argmax(dim=1, keepdim=True)
        accuracy += pred.eq(target.view_as(pred)).sum().item()
        ndata += len(data)

        t.set_postfix(loss=train_loss/ndata, err=100*(1-accuracy
            /ndata))

def eval_loss_and_error(loss, model, device, loader):
    # t0 = time.time()
    model.eval()
    center = model.get_or_build_center().to(device)
    center.eval()

    l, accuracy = np.zeros(model.y), np.zeros(model.y)
    center_loss, center_accuracy = 0., 0.
```

```
ensemble_loss, ensemble_accuracy = 0., 0.
ndata = 0

with torch.no_grad():
    for data, target in loader.single_loader():

        data, target = data.to(device), target.to(device)

        # single replicas
        outputs = model(data, split_input=False,
            concatenate_output=False)
        for a, output in enumerate(outputs):
            l[a] += loss(output, target, reduction='sum').
                item()
            pred = output.argmax(dim=1, keepdim=True)
            accuracy[a] += pred.eq(target.view_as(pred)).sum
                ().item()

        # ensemble
        output = torch.mean(torch.stack(outputs), 0)
        ensemble_loss += loss(output, target, reduction='sum
            ').item()
        pred = output.argmax(dim=1, keepdim=True)
        ensemble_accuracy += pred.eq(target.view_as(pred)).
            sum().item()

        # center
```

```
            output = center(data)
            center_loss += loss(output, target, reduction='sum')
                .item()
            pred = output.argmax(dim=1, keepdim=True)
            center_accuracy += pred.eq(target.view_as(pred)).sum
                ().item()
            ndata += len(data)



    l /= ndata
    accuracy /= ndata
    center_loss /= ndata
    center_accuracy /= ndata
    ensemble_loss /= ndata
    ensemble_accuracy /= ndata
    return l, (1-accuracy)*100, center_loss, (1-center_accuracy)
        *100,\
         ensemble_loss, (1-ensemble_accuracy)*100


@ex.config  # Configuration is defined through local variables.
def cfg():
    batch_size = 128     # input batch size for training
    epochs = 100         # number of epochs to train
    lr = 0.1             # learning rate
    weight_decay = 5e-4 # weight decay param (=L2 reg. Good
        value is 5e-4)
```

```
no_cuda = False      # disables CUDA training
nthreads = 2         # number of threads
save_model = False   # save current model to path
save_epoch = 10      # save every save_epoch model
keep_models = False  # keep all saved models
load_model = ""      # load model from path
droplr = 5           # learning rate drop factor (use 0 for
    no-drop)
opt = "adam"         # optimizer type
loss = "nll"         # classification loss [nll, mse]
model = "lenet"      # model type  [lenet, densenet,...]
dataset = "cifar10" # dataset  [mnist, fashion, cifar10,
    cifar100]
datapath = '~/data/'# folder containing the datasets (e.g.
    mnist will be in "data/MNIST")
logtime = 2          # report every logtime epochs
M = -1               # take only first M training examples
Mtest = -1           # take only first Mtest test examples
pclass = -1          # take only pclass training examples for
     each class
preprocess = False   # data normalization
gpu = 0             # gpu_id to use
dropout = 0          # dropout rate
save_zero_epoch = False
alpha =1             # multiplying softmax layer entries with
     alpha
# ROBUST ENSEMBLE SPECIFIC
```

```
    y=1                    # number of replicas
    use_center=False       # use a central replica
    g=1e-3                 # initial coupling value
    grate=1e-1             # coupling increase rate

    # Sensitivity Choice Parameters
    pooling = "max"        # pooling type
    activation_function = "relu" # activation function

    os.environ["CUDA_VISIBLE_DEVICES"] = to_gpuid_string(gpu)  #
        To be done before any call to torch.cuda



@ex.automain
def main(_run, _config):
    ## SOME BOOKKEEPING
    args = argparse.Namespace(**_config)
    print_config(_run); print()
    logdir = file_observer_dir(_run)
    if not logdir is None:
        from torch.utils.tensorboard import SummaryWriter
        writer = SummaryWriter(log_dir=f"{logdir}/{
            run_and_config_to_path(_run, _config)}")

    if args.save_model: # make temp file. In the end, the model
        will be stored by the observers.
```

```
    save_prefix = tempfile.mkdtemp() + "/model"

use_cuda = not args.no_cuda and torch.cuda.is_available()
print("USING␣CUDA␣=", use_cuda)
device = torch.device(f"cuda" if use_cuda else "cpu")

torch.manual_seed(args.seed)
torch.set_num_threads(args.nthreads)

## LOAD DATASET
loader_args = {'pin_memory': True} if use_cuda else {}
dtrain, dtest = get_dataset(args)
if args.pclass > 0:
    train_idxs = take_n_per_class(dtrain, args.pclass)
else:
    train_idxs = list(range(len(dtrain) if args.M <= 0 else
        args.M))

test_idxs = list(range(len(dtest) if args.Mtest <= 0 else
    args.Mtest))

print(f"DATASET␣{args.dataset}:␣{len(train_idxs)}␣Train␣and␣
    {len(test_idxs)}␣Test␣examples")

train_loader = RobustDataLoader(dtrain,
    y=args.y, concatenate=True,
    sampler=SubsetRandomSampler(train_idxs),
```

```
        batch_size=args.batch_size, **loader_args)
    test_loader = RobustDataLoader(dtest,
        y=args.y, concatenate=True,
        sampler=SubsetRandomSampler(test_idxs),
        batch_size=args.batch_size, **loader_args)

    ## BUILD MODEL
    Net = get_model_type(args)
    model = RobustNet(Net, y=args.y, g=args.g, grate=args.grate,
        use_center=args.use_center)
    model = model.to(device)

    if args.load_model:
        model.load_state_dict(torch.load(args.load_model + ".pt"
            ))
    ex.info["num_params"] = num_params(model)
    print(f"MODEL: {ex.info['num_params']} params")
    ## CREATE OPTIMIZER

    optimizer = get_optimizer(args, model)
    if args.droplr:
        gamma_sched = 1/args.droplr if args.droplr > 0 else 1
        scheduler = optim.lr_scheduler.MultiStepLR(optimizer,\
                milestones=[args.epochs//2, args.epochs*3//4,
                    args.epochs*15//16], gamma=gamma_sched)

    ## LOSS FUNCTION
```

```
loss = get_loss_function(args)

## REPORT CALLBACK
def report(epoch):
    model.eval()

    o = dict() # store scalar observations
    oo = dict() # store array observations
    o["epoch"] = epoch
    oo["train_loss"], oo["train_error"], o["
       train_center_loss"], o["train_center_error"],\
        o["train_ensemble_loss"], o["train_ensemble_error"]
           = \
            eval_loss_and_error(loss, model, device,
               train_loader)

    oo["test_loss"], oo["test_error"], o["test_center_loss"
       ], o["test_center_error"],\
        o["test_ensemble_loss"], o["test_ensemble_error"] =
           \
            eval_loss_and_error(loss, model, device,
               test_loader)

    o["coupl_loss"] = model.coupling_loss().item()

    oo["distances"] = np.sqrt(np.array([d.item()/model.
       num_params() for d in model.sqdistances()]))
```

```
        oo["norms"] = np.sqrt(np.array([sqn.item()/model.
            num_params() for sqn in model.sqnorms()]))
        o["gamma"] = model.g

        print("\n", pd.DataFrame({k:[o[k]] for k in o}), "\n")
        for k in o:
            ex.log_scalar(k, o[k], epoch)
            if logdir:
                writer.add_scalar(k, o[k], epoch)
        for k in oo:
            print(f"{k}:\t{oo[k]}")
            ex.log_scalar(k, np.mean(oo[k]), epoch) # Ref. https
                ://github.com/IDSIA/sacred/issues/465
            if logdir:
                writer.add_scalar(k, np.mean(oo[k]), epoch)
        print()

    ## START TRAINING
    report(0)
    if args.save_zero_epoch==True:
        epoch_str = '000'
        model_path = save_prefix+".pt"
        torch.save(model.state_dict(), model_path)
        if args.keep_models:
            kept_model_path = save_prefix+"_epoch_{}.pt".format(
                epoch_str)
            copyfile(model_path, kept_model_path)
```

```
            ex.add_artifact(kept_model_path, content_type="
                application/octet-stream")
        ex.add_artifact(model_path, content_type="application/
            octet-stream")

    for epoch in range(1, args.epochs + 1):
        epoch_str = str(epoch).zfill(3) #creates numbers like:
            001, 002, ..., 010
        print(epoch_str)
        train(loss, model, device, train_loader, optimizer)
        # torch.cuda.empty_cache()
        if epoch % args.logtime == 0:
            report(epoch)
        if epoch % args.save_epoch == 0 and args.save_model:
            model_path = save_prefix+".pt"
            torch.save(model.state_dict(), model_path)
            if args.keep_models:
                kept_model_path = save_prefix+"_epoch_{}.pt".
                    format(epoch_str)
                copyfile(model_path, kept_model_path)
                ex.add_artifact(kept_model_path, content_type="
                    application/octet-stream")
            ex.add_artifact(model_path, content_type="
                application/octet-stream")
        model.increase_g()
        if args.droplr:
            scheduler.step()
```

```
    # Save model after training
    if args.save_model:
        model_path = save_prefix+"_final.pt"
        torch.save(model.state_dict(), model_path)
        ex.add_artifact(model_path, content_type="application/
            octet-stream")
```

5.1.2 Estimating the Mean Dimension

```
### Estimates mean dimension of a neural network
# different output options (each node in the network, only last
    layer + softmay layer, on negative loglikelihood)



import torch
import argparse
import torch.optim as optim
from torch.utils.data import SubsetRandomSampler, DataLoader,
    ConcatDataset, Subset
import time
import os, datetime
from tqdm import tqdm
import pandas as pd
import numpy as np
import tempfile
import pdb
import pickle
```

```
#import matplotlib.pyplot as plt

from sacred import Experiment
from sacred.commands import print_config



ex = Experiment('RobustDNN')
# from sacred.utils import apply_backspaces_and_linefeeds # for
    progress bar captured output

# ex.captured_out_filter = apply_backspaces_and_linefeeds #
    doesn't work: sacred/issues/440
from sacred import SETTINGS

SETTINGS['CAPTURE_MODE'] = 'no'  # don't capture output (avoid
    progress bar clutter)

# my imports
from update_MD import update_MD
from sacreddnn.models.robust import RobustNet, RobustDataLoader
from sacreddnn.parse_args import get_dataset, get_loss_function,
     get_model_type, get_optimizer, Dataset
from sacreddnn.utils import num_params, l2_norm,
    run_and_config_to_path, \
     file_observer_dir, to_gpuid_string
```

```
@ex.config  # Configuration is defined through local variables.
def cfg():
    batch_size = 128     # input batch size for training
    #epochs = 100         # number of epochs to train
    #lr = 0.1             # learning rate
    #weight_decay = 5e-4 # weight decay param (=L2 reg. Good
        value is 5e-4)
    no_cuda = False      # disables CUDA training
    nthreads = 2         # number of threads
    #save_model = False  # save current model to path
    #save_epoch = 10     # save every save_epoch model
    #keep_models = False # keep all saved models
    load_model = ""      # load model from path
    #droplr = 5           # learning rate drop factor (use 0 for
        no-drop)
    #opt = "nesterov"    # optimizer type
    loss = "nll"         # classification loss [nll, mse]
    model = "lenet"      # model type  [lenet, densenet,...]
    dataset = "cifar10" # dataset  [mnist, fashion, cifar10,
        cifar100]
    datapath = '~/data/'# folder containing the datasets (e.g.
        mnist will be in "data/MNIST")
    #logtime = 2          # report every logtime epochs
    M = -1               # take only first M training examples
    Mtest = -1           # take only first Mtest test examples
    pclass = -1          # take only pclass training examples for
         each class
```

```
    preprocess = False  # data normalization
    gpu = 0             # gpu_id to use
    pca = True
    alpha = 1
    #last_layer =False
# ROBUST ENSEMBLE SPECIFIC
    y=1                 # number of replicas
    use_center=False    # use a central replica
    g=1e-3              # initial coupling value
    grate=1e-1          # coupling increase rate

    layer="nll"         # which node or output for mean
       dimension. ["nll", "last_layer_+_sm", "all_layer"]
    os.environ["CUDA_VISIBLE_DEVICES"] = to_gpuid_string(gpu)  #
        To be done before any call to torch.cuda



@ex.automain
def main(_run, _config):
    ## SOME BOOKKEEPING
    torch.set_flush_denormal(True)
    #torch.set_default_dtype(torch.double)
    args = argparse.Namespace(**_config)
    mydir = os.path.join(os.getcwd(), args.load_model[5:-3] +"_"
       + datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S'))
```

```
os.makedirs(mydir)
print_config(_run);



logdir = file_observer_dir(_run)
if not logdir is None:
    from torch.utils.tensorboard import SummaryWriter
    writer = SummaryWriter(log_dir=f"{logdir}/{
        run_and_config_to_path(_run, _config)}")

#if args.save_model:  # make temp file. In the end, the
    model will be stored by the observers.
#     save_path = tempfile.mkdtemp() + "/model.pt"

use_cuda = not args.no_cuda and torch.cuda.is_available()
print("USING CUDA =", use_cuda)
device = torch.device(f"cuda" if use_cuda else "cpu")

torch.manual_seed(args.seed)
torch.set_num_threads(args.nthreads)

## LOAD DATASET

loader_args = {'pin_memory': True} if use_cuda else {}
DATAPATH = '~/data/'
```

```
dtrain, dtest = get_dataset(args)

train_idxs = list(range(len(dtrain) if args.M <= 0 else args
    .M))

test_idxs = list(range(len(dtest)))

print(f"DATASET {args.dataset}: {len(train_idxs)} Train and
    {len(test_idxs)} Test examples")

train_loader = RobustDataLoader(dtrain,
                                y=args.y, concatenate=True,
                                sampler=SubsetRandomSampler(
                                    train_idxs),
                                batch_size=args.batch_size,
                                    **loader_args)
test_loader = RobustDataLoader(dtest,
                               y=args.y, concatenate=True,
                               sampler=SubsetRandomSampler(
                                   test_idxs),
                               batch_size=args.batch_size,
                                   **loader_args)
## BUILD MODEL
Net = get_model_type(args)
model = RobustNet(Net, y=args.y, g=args.g, grate=args.grate,
    use_center=args.use_center)
model = model.to(device)
```

```
if args.load_model:
    model.load_state_dict(torch.load(args.load_model,
        map_location='cpu'))

ex.info["num_params"] = num_params(model)
print(f"MODEL: {ex.info['num_params']} params")

## LOSS FUNCTION
loss = get_loss_function(args)

center = model.get_or_build_center().to(device)
center = center.eval()
loss_entr = torch.nn.CrossEntropyLoss(reduction='none')
sm = torch.nn.Softmax(dim=1)
alpha =args.alpha



model = center
# choose outputs for which to estimate mean dimension
agg={}
num=1
if args.layer=='all_layer':
    for i in model.children():
        layer_name = "layer_" + str(num)
        # agg: tuple; (tuple of (count, mean, M2) of
            activations/nodes in mod , tuple of (count, mean,
```

```
                M2) of finite changes for each input of nodes in
                mod)
            agg[layer_name]=((0,0,0),(0,0,0))
            num +=1
        agg['nll']=((0,0,0),(0,0,0))
        agg['sm']=((0,0,0),(0,0,0))


if args.layer=='last_layer_+_sm':
        agg['last_layer']=((0,0,0),(0,0,0))
        agg['sm']=((0,0,0),(0,0,0))

if args.layer=='nll':
        agg['nll']=((0,0,0),(0,0,0))



# get all data
dataiter = iter(test_loader)

input = torch.autograd.Variable(torch.FloatTensor()).to(
    device)
output = torch.autograd.Variable(torch.LongTensor()).to(
    device)
for data, target in test_loader.single_loader():
        data, target = data.to(device), target.to(device)
```

```
        input = torch.cat((input, data.detach()))
        output = torch.cat((output, target.detach()))


    data = input.detach()
    target = output.detach()
    #print("data shape " + str(data.shape))
    N = data.shape[0]  # number of inputs
    # transforming to 2d matrix
    xmat_all = data.view(N,-1).detach()

    # load data summary for pca transformation
    data_mean = torch.from_numpy(np.load("mean.npy")).to(device)
    data_std = torch.from_numpy(np.load("std.npy")).to(device)
    data_comp = torch.from_numpy(np.load("comps.npy")).to(device
        )

    # pca transformation of data
    if args.pca:
    # transforming to 2d matrix
        xmat_all = data.view(N,-1).detach()

        xmat_all =((xmat_all - data_mean)/data_std).detach()

        xmat_all = torch.matmul(xmat_all, data_comp.transpose
            (0,1)).detach()
```

```
k = xmat_all.shape[1]  # (images.shape[2]) * (images.shape
    [3])  # vector length of all features/inputs/pixel
pca=args.pca
batch_size = args.batch_size
x = xmat_all
y = target
# run mean dimension estimation (collecting finite
    differences)
with torch.no_grad():
    for j in range(N-1): # N=number of samples
        agg= update_MD(x[j], x[j+1], y[j], y[j+1], model,
            pca, loss_entr, sm, args, data_mean, data_std,
            data_comp, device, batch_size, agg, args.layer)


    def finalize(existingAggregate):
        (count, mean, M2) = existingAggregate
        if count < 2:
            return float("nan")
        else:
            (mean, variance, sampleVariance) = (mean, M2 /
                count, M2 / (count - 1))
            return (mean, variance, sampleVariance)

    MD={}
    i=0
```

```
        # finalize mean dimension computation by using variance
            of output and variance of finite differences
        for mod in agg.keys():

            agg[mod]=(finalize(agg[mod][0]),finalize(agg[mod
                ][1]))

            MD[str(mod)]=(torch.sum(agg[mod][1][1],0)/(2*agg[mod
                ][0][1])).cpu()
            i+=1

        # save mean dimension
        MD_file = open(mydir+"/MD_train.pkl","wb")
        pickle.dump(MD, MD_file)
        MD_file.close()
```

Required 'Update Function'

```
# function that computes finite differences for each feature for
    2 input (x and x1)
import numpy as np
import torch
import pdb
from sacreddnn.parse_args import Dataset
from torch.utils.data import DataLoader #, Subset


def update_MD(x, x1, label, label1, center, pca, loss_entr, sm,
```

```
args, data_mean, data_std, data_comp, device, batch_size, agg
, layer):

 model=center
 dx = x1 - x

 #print(torch.get_default_dtype())

 n = x.shape[0] #number of parameters

 ###################### generating helper matrix u
 u1 =torch.zeros(n,1).to(device)
 u2 =torch.ones(n,1).to(device)
 u3= torch.diag(torch.ones(n,)).to(device)
 u4 = abs(u3-1).to(device)
 u= torch.cat((u1,u2,u3,u4), dim=1)



 # generating matrix DX: colums represent vectorized (changed
    ) images
 k = u.shape[1] #width of matrix (DX) # should be 2n+2

 ddx= (dx.repeat(k,1)).transpose(0,1).detach()


DX= (torch.mul(ddx,u)+ (x.repeat(k,1)).transpose(0,1)).
```

```
    detach()



# transpose to have flattened images in rows
DX = DX.transpose(0,1).detach()
# labels for each image
true_label = torch.cat((label.repeat(1), label1.repeat(1),
    label.repeat(n), label1.repeat(n)),0).detach()



#PCA-inverse transformation
if pca==True:
    DX= torch.mm(DX, torch.inverse(data_comp.transpose(0,1))
        ).detach()
    DX= (torch.mul(DX, data_std) + data_mean).detach()

#transforming data to required form
dx_view = DX.view((-1, 3, 32, 32))
#create dataset
imgs =Dataset(dx_view, torch.ones(10000))
act={}



for mod in agg.keys():
```

```
        act[mod]=[]

    # choosing proper output choice
    if layer=='all_layer':
        l=1
        for mod in agg.keys():
            for img, lbl in DataLoader(imgs, batch_size):
                img1 = img.to(device)
                # act is a dict where each key has a value that
                    contains a list of node activations (
                    dimension depends on layer/mod)
                act[mod].append(model[0:l](img1).detach())
            l+=1
    if layer=='nll':
        for img, lbl in DataLoader(imgs, batch_size):
            img1 = img.to(device)
            # act is a dict where each key has a value that
                contains a list of node activations (dimension
                depends on layer/mod)
            act['nll'].append(model(img1).detach())

    if layer=='last_layer_+_sm':
        for img, lbl in DataLoader(imgs, batch_size):
            img1 = img.to(device)
            # act is a dict where each key has a value that
                contains a list of node activations (dimension
                depends on layer/mod)
```

```
            act['last_layer'].append(model(img1).detach())

    #with torch.no_grad():


    #updating mean and variance of finite changes (thus we dont
        need to save all data; only these aggregates)
    def update(existingAggregate, newValue):
        (count, mean, M2) = existingAggregate
        count += 1
        delta = newValue - mean
        mean += delta / count
        delta2 = newValue - mean
        M2 += delta * delta2
        return (count, mean, M2)



    if layer=='nll':

        # getting finite changes from nll
        act_temp=torch.cat(act['nll']).detach()
        nll_temp=loss_entr(act_temp, true_label)
        finite_changes = nll_temp[2:(n+2)]-nll_temp[0]
        agg['nll']=(update(agg['nll'][0], nll_temp[0]),update(
            agg['nll'][1], finite_changes))
```

```
if layer=='last_layer_+_sm':


    # getting finite changes from last layer scores
    act_temp=torch.cat(act['last_layer']).detach()

    act['last_layer']=(act_temp[0,:],act_temp[2:(n+2),:]-
        act_temp[0,:] )

    # agg: tuple; (tuple of (count, mean, M2) of activations
        /nodes in mod , tuple of (count, mean, M2) of finite
        changes for each input of nodes in mod)
    agg['last_layer']=(update(agg['last_layer'][0], act['
        last_layer'][0]),update(agg['last_layer'][1], act['
        last_layer'][1]))

    # getting finite changes from class probabilities
    sm_temp=torch.nn.functional.softmax(act_temp,dim=1)
    act['sm']=(sm_temp[0,:],sm_temp[2:(n+2),:]-sm_temp[0,:]
        )

    # agg: tuple; (tuple of (count, mean, M2) of activations
        /nodes in mod , tuple of (count, mean, M2) of finite
        changes for each input of nodes in mod)
```

```
        agg['sm']=(update(agg['sm'][0], act['sm'][0]),update(agg
            ['sm'][1], act['sm'][1]))




i=0
if layer=='all_layer':
    # only for for modules in model (without softmax and nll
        ):
    modules_list= list(agg.keys())[:-2]
    for mod in modules_list:

        # cat of the list (values of dict are lists)
        act[mod]=torch.cat(act[mod]).detach()
        # act: tuple of dim 2; (nodes/activations in layer/
            mod, finite changes for each input of nodes in
            mod)
        act[mod]=(act[mod][0,:],act[mod][2:(n+2),:]-act[mod
            ][0,:] )

        # agg: tuple; (tuple of (count, mean, M2) of
            activations/nodes in mod , tuple of (count, mean,
             M2) of finite changes for each input of nodes in
             mod)
        agg[mod]=(update(agg[mod][0], act[mod][0]),update(
```

```
                agg[mod][1], act[mod][1]))


        # getting finite changes from nll
        act_temp=torch.cat(act['nll']).detach()
        nll_temp=loss_entr(act_temp, true_label)
        finite_changes = nll_temp[2:(n+2)]-nll_temp[0]
        agg['nll']=(update(agg['nll'][0], nll_temp[0]),update(
            agg['nll'][1], finite_changes))

        # getting finite changes from class probabilities
        sm_temp=torch.nn.functional.softmax(act_temp,dim=1)
        act['sm']=(act_temp[0,:],act_temp[2:(n+2),:]-act_temp
            [0,:] )

        # agg: tuple; (tuple of (count, mean, M2) of activations
            /nodes in mod , tuple of (count, mean, M2) of finite
            changes for each input of nodes in mod)
        agg['sm']=(update(agg['sm'][0], act['sm'][0]),update(agg
            ['sm'][1], act['sm'][1]))


    return(agg)
```