UNIVERSITY OF CALIFORNIA, MERCED

**Approaches to Interpret Deep Neural Networks**

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering & Computer Science

by

Suryabhan Singh Hada

Committee in charge:

Professor Miguel Á. Carreira-Perpiñán, Chair
Professor Ming-Hsuan Yang
Professor Shawn Newsam

2022

The dissertation of Suryabhan Singh Hada is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

(Professor Ming-Hsuan Yang)

(Professor Shawn Newsam)

(Professor Miguel Á. Carreira-Perpiñán, Chair)

University of California, Merced

2022

DEDICATION

To my parents: Mahendra Singh Hada and Prem Kanwar.

TABLE OF CONTENTS


Signature Page . . . iii

Dedication . . . iv

Table of Contents . . . v

List of Figures . . . vii

List of Tables . . . viii

Acknowledgements . . . ix

Vita and Publications . . . x

Abstract . . . xii

Chapter 1 Introduction . . . 1

Chapter 2 Related Work . . . 4
2.1 Inspecting the structure of a deep neural network . . . 4
2.2 Explaining outcome of an input . . . 8
2.2.1 Explanation by an interpretable model . . . 8
2.2.2 Explanations by examples . . . 10
2.2.3 Explanations by saliency maps . . . 11
2.3 Counterfactual explanations . . . 13

Chapter 3 Sampling the "Inverse Set" of a Neuron . . . 17
3.1 The inverse set of a neuron: definition . . . 17
3.2 Sampling the inverse set: an optimization approach . . . 18
3.3 Sampling in feasible region . . . 20
3.3.1 Intersection of inverse sets . . . 22
3.4 Experiments . . . 23
3.5 Conclusion . . . 27

Chapter 4 Interpreting Deep Neural Networks Using Sparse Oblique Trees 28
4.1 Mimicking the network using sparse oblique decision trees . 30
4.2 Manipulating the deep net features to alter its classification 30
4.3 Experiments . . . 33

4.3.1 Results on VGG16 . . . . . 34
4.3.2 Results on LeNet5 . . . . . 42
4.4 Conclusion . . . . . 45

Chapter 5 Interpretable Image Classification Using Sparse Oblique Trees . 47
5.1 Proposed approach . . . . . 48
5.1.1 Interpreting weights of the decision nodes . . . . . 49
5.2 Experiments . . . . . 50
5.2.1 Datasets description . . . . . 50
5.2.2 Experiment setup . . . . . 52
5.2.3 Interpretability results on Fashion-MNIST . . . . . 53
5.2.4 Sub-groups in sneakers and pullover class . . . . . 55
5.2.5 Difference between pairs of classes . . . . . 56
5.2.6 Feature selection for a given instance . . . . . 58
5.3 Conclusion . . . . . 60

Chapter 6 Conclusion and Future Work . . . . . 61

Appendix A Neural Network Architectures . . . . . 63

LIST OF FIGURES

Figure 3.1: Samples generated for Volcano class in the CaffeNet. . . . . 21
Figure 3.2: Comparison with PPGN approach. . . . . 23
Figure 3.3: Inverse set for hidden neurons in CaffeNet. . . . . 24
Figure 3.4: Inverse set for MIT Places dataset. . . . . 25
Figure 3.5: Inverse set intersection. . . . . 25
Figure 3.6: Inverse set intersection with a hidden neuron. . . . . 26

Figure 4.1: Mimicking part of a neural net with a decision tree. . . . . 29
Figure 4.2: Masking operation in the deep neural network. . . . . 32
Figure 4.3: Tree mimic's error and size plots for VGG16. . . . . 35
Figure 4.4: Tree selected as mimic for VGG16 features ($\lambda = 1$). . . . . 36
Figure 4.5: Tree mimic for VGG16 features with one leaf per class ($\lambda = 33$). 37
Figure 4.6: Confusion matrices for VGG16 (test set). . . . . 39
Figure 4.7: Confusion matrices for VGG16 (training set). . . . . 40
Figure 4.8: Illustration of masks for a particular image. . . . . 41
Figure 4.9: Tree mimic's error and size plots for LeNet5. . . . . 42
Figure 4.10: Tree selected as mimic for LeNet5 features ($\lambda = 20$). . . . . 43
Figure 4.11: Confusion matrices for LeNet5 (test set). . . . . 44
Figure 4.12: Confusion matrices for LeNet5 (training set). . . . . 45

Figure 5.1: Single instance of each class in Fashion-MNIST dataset. . . . . 50
Figure 5.2: Trees' error and size plots for Fashion-MNIST dataset. . . . . 52
Figure 5.3: Tree selected to interpret the Fashion-MNIST dataset. . . . . 54
Figure 5.4: Class histograms for the tree trained on Fashion-MNIST dataset. 54
Figure 5.5: CART tree trained on Fashion-MNIST dataset. . . . . 55
Figure 5.6: Features separating Dress and Coat class. . . . . 56
Figure 5.7: Features separating group of classes. . . . . 57
Figure 5.8: Feature selection for a given instance. . . . . 59

LIST OF TABLES

Table 3.1: Numeric comparison between the proposed approach and PPGN. . 24

Table 4.1: Classes in our ImageNet subset. . . . . 34

Table 5.1: Features selected by the sparse oblique tree for different dataset. . 51
Table 5.2: Training and test errors for CART and TAO on different datasets. 53

Table A.1: Architecture of LeNet5. . . . . 63
Table A.2: Architecture of our modified VGG16. . . . . 64

ACKNOWLEDGEMENTS

This dissertation could not have been possible without the support of some amazing people in my life and I am sincerely grateful to them.

First and foremost, I want to thank my advisor Dr. Miguel Á. Carreira-Perpiñán, who gave me the opportunity to start my research career in his lab. I am thankful for his valuable guidance throughout my Ph.D., which helped me overcome problems in my research work. I also want to thank my committee members, Dr. Ming-Hsuan Yang and Dr. Shawn Newsam, for their valuable suggestions on this dissertation.

During my time at UC Merced, I have met some great people who made my time in Merced a memorable experience. Thank you for being great friends, Sai, David, Anupam, Ahmad, Suhani, Ashish, and many more. Especially Sai and Anupman, thanks for taking time off and planning multiple trips to different national parks. It was always a very relaxing experience. I also want to thank my labmates Yerlan, Arman, and Mazghan for being very cooperative peers. I want to give a special thank to Jordan for many great memories and for encouraging me whenever I felt down.

Next, I want to thank my friends from India: Akshay Nagpurkar, Parth Agrawal , Pulkit Garg, Vivek Kumar, Sumit Bhanwala, Shivam Prasad, Vineet Sharma, and many more. Without friends like you, I would never have had the courage to quit my job and pursue my Ph.D. I also want to thank my childhood friends Devendra and Shefali, who always encouraged me to follow my research career and were always ready to talk to me even with a twelve-hour time difference.

Most importantly, I want to thank my family, without whom I could not have achieved this goal in my life. My sisters: Usha, Renu, and Pritam, had a huge impact on my life, introducing me to science as a kid. Finally, my deepest appreciation goes to my parents. They always made sure that I always got a good education and did everything to create an environment where I could study comfortably without any worry. I will always be indebted to my parents, and this dissertation is dedicated to them.

## VITA

| | |
|---|---|
| 2014 | Integrated Masters of Technology (Bachelor & Masters) in Mathematics and Computing, Indian Institute of Technology (B.H.U.), Varanasi, India |
| 2022 | Ph.D. in Electrical Engineering and Computer Science, University of California, Merced |

## PUBLICATIONS

Miguel Á. Carreira-Perpiñán and Suryabhan Singh Hada: "More Interpretable Decision Trees: Pruning via Node Descent and the Delta Penalty" *in submission (2022).*

Miguel Á. Carreira-Perpiñán and Suryabhan Singh Hada: "Inverse classification with logistic and softmax classifiers: efficient optimization" *in submission (2022).*

Miguel Á. Carreira-Perpiñán and Suryabhan Singh Hada: "Very Fast, Approximate Counterfactual Explanations for Decision Forests" *in submission (2022).*

Suryabhan Singh Hada and Miguel Á. Carreira-Perpiñán: "Interpretable Image Classification using Sparse Oblique Decision Trees" in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2022)*, pp. 2759–2763.

Suryabhan Singh Hada, Miguel Á. Carreira-Perpiñán and Arman Zharmagambetov: "Sparse Oblique Decision Trees: A Tool to Understand and Manipulate Neural Net Features" in *Data Mining and Knowledge Discovery (dmkd 2022).*

Suryabhan Singh Hada, Miguel Á. Carreira-Perpiñán and Arman Zharmagambetov: "Understanding and Manipulating Neural Net Features Using Sparse Oblique Classification Trees" in *International Conference on Image Processing (ICIP 2021)*, pp. 3707–3711.

Suryabhan Singh Hada and Miguel Á. Carreira-Perpiñán: "Exploring Counterfactual Explanations for Classification and Regression Trees" in *ECML PKDD 3rd International Workshop and Tutorial on eXplainable Knowledge Discovery in Data Mining (XKDD 2021)*, pp. 489—504.

Suryabhan Singh Hada and Miguel Á. Carreira-Perpiñán: "Sampling the "Inverse Set" of a Neuron: An Approach to Understanding Neural Nets" in *International Conference on Image Processing (ICIP 2021)*, pp. 3712–3716.

Suryabhan Singh Hada and Miguel Á. Carreira-Perpiñán: "Style Transfer by Rigid Alignment in Neural Net Feature Space" in *Winter Conference of Applications on Computer Vision (WACV 2021)*, pp. 2575–2584.

Suryabhan Singh Hada, Miguel Á. Carreira-Perpiñán and Arman Zharmagambetov: "Sparse Oblique Decision Trees: A Tool to Understand and Manipulate Neural Net Features" in Unpublished manuscript, April 07,2021, arXiv:2104.02922.

Miguel Á. Carreira-Perpiñán and Suryabhan Singh Hada: "Counterfactual Explanations for Oblique Decision Trees: Exact, Efficient Algorithms" in *35th AAAI conference on Artificial Intelligence (AAAI 2021)*, pp. 6903–6911.

Arman Zharmagambetov, Suryabhan Singh Hada, Magzhan Gabidolla and Miguel Á. Carreira-Perpiñán: "Non-Greedy Algorithms for Decision Tree Optimization: An Experimental Comparison" in *International Joint Conference on Neural Networks (IJCNN 2021)*.

ABSTRACT OF THE DISSERTATION

**Approaches to Interpret Deep Neural Networks**

by

Suryabhan Singh Hada
Doctor of Philosophy in Electrical Engineering & Computer Science
University of California Merced, 2022
Professor Miguel Á. Carreira-Perpiñán, Chair

Practical deployment of deep neural networks has become widespread in the last decade due to their ability to provide simple, intelligent, and automated processing of the tasks that up to now were hard for other machine learning models. There is an enormous financial and societal interest in deep neural networks as a viable solution for many practical problems such as computer vision, language processing, financial fraud detection, and many more. At the same time, some concerns regarding the safety and ethical use of these models have arisen as well. One of the main concerns is interpretability, i.e., explaining how the model makes a decision for an input. Interpretability is one of the most important problems to address for building trust and accountability as the adoption of deep neural networks has increased significantly in sensitive areas like medicine, security, and finance.

This dissertation proposes two novel approaches to interpreting deep neural networks. The first approach focuses on understanding what information is retained by the neurons of a deep net. We propose an approach to characterize the region of input space that excites a given neuron to a certain level. Inspection of these regions by a human can reveal regularities that help to understand the neuron.

In the second approach, we provide a systematic way to understand what group

of neurons in a deep net are responsible for a particular class. This allows us to study the relation between deep net features (neuron's activation) and output classes; and how different classes are distributed in the latent space. We also show that out of thousands of neurons in the deep net, only a small subset of neurons is associated with a specific class.

Finally, we demonstrate that the latter approach can also be used to interpret large datasets. This is achieved by applying the second approach directly over the input features. This allows us to understand what input features are related to a specific class and what set of features differentiates between a group of classes or even sub-groups within a given class.

# Chapter 1

# Introduction

In the last few years, deep neural nets have become the preferred model in a number of practical problems, such as computer vision, speech and language processing, games, self-driving cars, and other engineering applications; legal, financial, and medical applications; and many others among non-engineering applications. Yet, the way neural nets are defined and optimized, and the sheer size and complexity of state-of-the-art deep nets, make them very hard to understand in explanatory terms. This problem is also true for other machine-learning models, but deep nets are more complicated than any other black-box model. Deep nets can learn a complicated correspondence between complex high-dimensional inputs and outputs via numerical optimization techniques that minimize a prediction loss over a large, labeled dataset. This makes them highly accurate in terms of predictive performance but remarkably difficult to understand in terms of how exactly they come up with a prediction for an input. We do not know what a given neuron represents, how a group of neurons interact with each other; or what happens if we remove some of them; or what we should change in the model to correct a wrong prediction for a given input; or what information is retained by the network. Further, for reasons not well understood, the deep net decisions are very sensitive to minute alterations

of the input in ways that can be used adversarially.

Another issue that makes these models working opaque is the data on which these models are trained. We can summarize small tabular datasets, but understanding the bigger datasets is challenging. For example, in image datasets, it is difficult to understand a given class's basic concepts. Since there is so much irrelevant information in an image, it is hard to understand what part of the image is important or what common concept defines a particular category of the class.

Most of these questions are not new [32, 73], but they have become urgent due to the widespread deployment of deep nets in sensitive applications. Indeed, this led to the creation of legislation to monitor the use of AI systems and data, such as the EU General Data Protection Regulation (GDPR) [31, 89]. Occasional wrong recommendation of a product is not serious, but crafting an adversarial attack so that the AI system does not recommend a product deliberately, is serious. This is more serious in the fields where the model error or attacks can be fatal, such as finance, medical field, or autonomous driving. For example, an adversarial attack on a deep net doing risk analysis in a trade can lead to a large financial loss; or an error detecting early signs of cancer can cause life-threatening situations. Therefore, there is a great need to understand the working of AI systems and provide explanations that are not limited to researchers but also to the end-users.

The emphasis of this dissertation is on understanding what information is stored in a given deep neural network's parameters and what internal features computed by this network are responsible for a particular class. This dissertation is organized as follows:

- In chapter 2, we present an extensive literature review of deep neural networks' interpretability. This covers the methods that inspect the structure of a deep net (section 2.1), providing explanation for a given input instances (section 2.2), and counterfactual explanations (section 2.3).

- In chapter 3, we introduce a new approach to explain the nature of individual neurons in the deep net. We achieve this by characterizing the region of input space that excites a given neuron to a certain level.

- In chapter 4, we propose a novel approach to understand the relation between deep net features (neuron's activation) and output classes, using sparse oblique decision trees. This provides a global explanation of the entire network, which is independent of any input data.

- In chapter 5, we extend the idea of connecting deep net features and classes to interpret image datasets using sparse oblique decision trees.

- In chapter 6, we conclude this dissertation, and provides future research directions.

# Chapter 2

# Related Work

In this chapter, we provide an extensive overview of deep neural networks' interpretability literature. As this is a very active research area, this is not an exhaustive literature review but covers important works widely discussed in the interpretability literature.

## 2.1 Inspecting the structure of a deep neural network

There are two major approaches to inspect the structure of a deep neural network (DNN): feature inversion and activation maximization.

The primary goal of feature inversion is to understand what information is retained by each layer of the network. The basic idea is to project the features from a layer to the input space so they can be visualized. Mahendran and Vedaldi [54] formulate the feature inversion problem as an optimization problem. The objective function is to minimize the Euclidean distance between the original image features and the generated image features. The authors point out that generated images are noisy, and to mitigate this, they use total variation as a regularizer during the op-

timization. A similar approach is used by Wei et al. [90], but they use data-driven patch priors as regularizers to improve the quality of generated images. Dosovitskiy and Brox [24] take a different approach: they train a deep neural network to generate images from the features. Xia et al. [94] extend the work of Mahendran and Vedaldi [54] to visualize what every filter has learned in the network. Authors show that even for distinct features, a given filter has the same texture and color. They also show that images generated with shallow layer filters have monotonous colors and the local structure is simple. On the other hand, as layers get deeper, the generated images' colors become plentiful and the local structures become more intricate.

On the other hand, activation maximization involves finding an image that maximize the response of a neuron. The problem can be formulated as:

$$\arg\max_{\mathbf{x}} f(\mathbf{x}) + \mathcal{R}(\mathbf{x}) \tag{2.1}$$

where real-valued function $f$ gives the activation value of the given neuron for an input image $\mathbf{x}$. $\mathcal{R}$ is the regularizer term to ensure the generated image is human interpretable. Activation maximization is done by taking a random image and then back-propagating it through the network to maximize the activation of the neuron of interest. This same formulation is used in Erhan et al. [26], Simonyan et al. [78], Nguyen et al. [63], Nguyen et al. [64] and many more.

Simonyan et al. [78] use activation maximization to visualize the second last layer neurons that represent neurons specific to a class. In this case, $f$ represents unnormalized class scores rather than class posteriors (last layer). The reason for that is the maximization of the class posterior can be achieved by minimizing the scores of other classes, which would not produce any useful information about the neuron. Although authors propose their approach only for the neurons in the last layer, it is the same for any neuron in the network.

Zhou et al. [108] restrict their search to the training data. For a given neuron, they sample around 200K images and select the images that have top k activation.

Next, for an image, they occlude image segments one by one to measure the change in the activation. They repeat this process for all the images selected in the first step. Image segments with large activation changes are selected and summarized to explain the concept learned by the neuron.

Yosinski et al. [98] propose two tools for visualizing and interpreting deep neural networks. The first tool visualizes the change in activation of a neuron in response to user input. The authors claim that analyzing the live activation and observing as they change in correspondence of different inputs helps to explain the DNN's behavior. The second tool is feature inversion, similar to Mahendran and Vedaldi [54], the only difference is that they use Gaussian blur as the regularizer.

Although these approaches are good, they do not consider the fact that there are multiple images that can maximally activate the same neuron. Nguyen et al. [64] try to address this problem by initializing the input image with the mean of a cluster of training images for activation maximization. The authors then repeat this process to generate different images using activation maximization. They call it *Multifaceted feature visualization*. However, the generated images are very noisy and very little diverse.

Both feature inversion and activation maximization suffer from a major problem: the result is a very noisy image or a fooling image [62, 83], which mostly makes little sense to humans. As mentioned above, to tackle this problem, many handcrafted regularizers are used like, $\alpha$-norm [78], Gaussian blur [98], total variation [54], jitter [59], data-driven patch priors [90], and center-biased regularization [64]. All these methods have helped to improve the quality of the generated images, but it is still not sufficient to create natural-looking images.

Dosovitskiy and Brox [25] propose a method to train a generative network based on generative adversarial networks [30], to generate natural-looking images from a feature vector. Nguyen et al. [63] incorporate this generative network into activation maximization. The authors use the same objective function as Simonyan et al. [78],

but instead of directly optimizing over images space, they optimize over features space. Also, they use the generative network from Dosovitskiy and Brox [25] to generate the final image. Compared to all previous methods, generated images are more natural-looking and human-understandable. Later, Nguyen et al. [65] use their *Plug and Play Generative Networks (PPGN)*, to generate more natural-looking images by introducing an additional prior via a denoising autoencoder in the feature space. Moreover, they also improve upon Nguyen et al. [64], generating comparatively more diverse images for a single neuron. For this, they define a probability distribution over the generative network and denoising encoder. Images are generated by sampling iteratively over this distribution using Markov chain Monte Carlo (MCMC) sampling.

Finally, there also exist approaches that are not based on optimization [6, 60]. Bau et al. [6] instead of generating images that maximally activate a neuron, use the neurons' ability to segment an image over a range of visual concepts that include not only high-level concepts like objects but also low-level information such as texture and colors. Their approach is to use a dataset containing images where objects are annotated along with color and texture information for each pixel. These images act as ground truth, and then neuron activation maps are used to segment these images. The segments with higher IoU (intersection over union) are picked as the concept captured by the neuron. Mu and Andreas [60] extend this work by combining the multiple different atomic concepts associated with a given neuron using logical expressions. They achieve this by iteratively constructing logical formulas from atomic concepts via beam search with a fixed beam size.

## 2.2 Explaining outcome of an input

### 2.2.1 Explanation by an interpretable model

In this section, we discuss methods that decompose the network prediction in terms of more interpretable models like decision trees, boolean rules, or linear models. The topic of extracting sets of rules from neural nets was actively researched in the 1990s [3, 32, 57]. Two basic approaches were used: in rule extraction as search [29, 86], a specialized heuristic search over possible rules was based on the neurons' connectivity pattern, but this assumed binary activations and did not scale beyond small networks. In rule extraction as learning, or teacher-student approach [17, 18, 23], one trains a decision tree to mimic the neural net by using the latter's predictions on the training set (possibly augmented with random instances).

Craven and Shavlik [17] propose an approach to extract rules from the network in the form of DNF (disjunctive normal form) expressions. For a given class, this DNF expression comprises the input features. For an input of a given class, if the corresponding boolean expression does not yield "true", then a new term is added to the expression to make it "true". This process is repeated until the DNF expression results "true" for all the training instances. To extract the rule from the network, Craven and Shavlik [18] train a decision tree from the neural network based on the teacher-student approach. The authors use maximizing the fidelity to the network as an objective to grow the tree. Training data has also been perturbed randomly to increase the size of the training set. This helps in improving the performance of the tree, especially at the nodes close to the leaf, where the number of instances is less. Once the tree is trained, rules can be extracted from the tree in the form of "if-else" statements.

The fundamental problem with these methods is that traditional decision tree learning algorithms such as CART [9] or C4.5 [70] are unable to learn small yet accurate enough trees to be useful mimics of a neural net except in very small prob-

lems. Moreover, these methods require data to be discretized to create the rules, which may be easy for small tabular data, but for big datasets like images, it is very difficult. The size of these rules is usually proportional to the number of the input features [17], which is very large in datasets like images.

To provide a global explanation, Thiagarajan et al. [84] divides the feature space into K (user-defined) overlapping factors. Then it creates meta-features to define each cluster and label them using a random forest. Finally, it trains a decision tree over these meta-features to approximate the neural network. Similar to Craven and Shavlik [17], it also cannot handle large datasets. Ribeiro et al. [71] introduce *Local Interpretable Model-agnostic Explanations (LIME)*, an approach to provide a local explanation of the network prediction for an input. The key idea is to approximate the complex model with an interpretable model locally around the prediction. LIME samples data in the neighborhood of the given input by perturbing it (eliminating image segments or zeroing some features). Then, it fits a sparse linear model over those data points. This local model provides the importance of features in the network prediction for the input. Ribeiro et al. [72] is an extension of Ribeiro et al. [71]. It creates an explanation in the form of decision rules (in terms of features) that are tied to the prediction, such that the change in the rest of the features does not affect the black box prediction. They call these features *Anchors*. They use a bandit algorithm that randomly generates anchors with the maximum coverage and user-defined threshold for model prediction confidence.

Murdoch et al. [61] propose a method to give importance to each word in a sentence for LSTMs [42]. It linearizes the activation functions that enable them to decompose the LSTM forward pass to provide two contributions: a sentence without the word and only the word alone. Singh et al. [79] extends this idea to be used for any type of deep network. Using feature's importance scores, they also provide an approach based on agglomerative clustering to show how different features cluster to affect the decision of the network. They call it *Agglomerative contextual*

*decomposition (ACD).*

Zhang et al. [102] use agglomerative clustering to explain how the network uses semantic information to classify images belonging to a given class. For this, the authors first modify the network architecture to have disentangled filters in the high convolution layers: each filter belongs to specific semantic information. They also make sure that each filter is consistently activated by the same object region over different input images. Now, for a given class, they create a tree using agglomerative clustering, where each node contains a weight vector. This weight vector is used to define the contribution of each filter in the classification of an image. Leaf nodes contain information specific to a single image, and as nodes move up, it contains filter information common to a group of images.

Another way to seek input features that are particularly important in predicting a given instance's class are *Shapley values*, originally proposed to attribute reward among players of a cooperative game [75]. Calculating Shapley values is NP-hard, so they are approximated in practice [20, 53, 58, 81]. The ability of Shapley values to provide explanations that are useful for humans has also been criticized [49].

### 2.2.2 Explanations by examples

To explain deep neural network prediction for a test input, these methods provide a set of inputs that influence neural network prediction. Koh and Liang [47] present *Influence functions* that describe this influence in terms of change in the test prediction, when a training example is removed from the dataset. Suppose $f(\mathbf{x}_t, \Theta)$ is the model error for a given test sample $\mathbf{x}_t$, with optimal parameters $\Theta$; and $f(\mathbf{x}_t, \Theta_{-z})$ is the model prediction for $\mathbf{x}_t$ when the training sample $\mathbf{z}$ is removed; thus model has new parameters $\Theta_{-\mathbf{z}}$. The influence is defined as the change in the prediction of $\mathbf{x}_t$ that is: $f(\mathbf{x}_t, \Theta) - f(\mathbf{x}_t, \Theta_{-\mathbf{z}})$. The authors propose a second-order optimization technique to approximate this difference without training the model again. Koh et al.

[48] investigate the accuracy of Influence functions by removing a large number of data points. They show that the Influence functions correlate well with the change in the performance of the model after removing the data points.

Although, *Influence functions* approach is good, Yeh et al. [97] showed that as the dataset size grows, Influence function computation becomes time consuming. It also runs into numerical issues in practice. For instance, producing no influence for a test input from all the training points.

Yeh et al. [97] represent the influence as a weighted linear combination of the network feature $\mathbf{f}_i$ with weights $\alpha_i$. The authors, show that with certain architecture modifications, the network prediction for a given test input (with $\mathbf{f}_{\text{test}}$ as feature) is equal to $\sum_i^n \alpha_i \mathbf{f}_i \mathbf{f}_{\text{test}}$. Here, $\mathbf{f}_i \mathbf{f}_{\text{test}}$ gives the similarity between $\mathbf{f}_i$ (training feature) and $\mathbf{f}_{\text{test}}$ (test feature); and $\alpha_i$ gives the weight to this similarity. By measuring both $\mathbf{f}_i \mathbf{f}_{\text{test}}$ and $\alpha_i$, the influence of a training example on the network prediction can be calculated. Recent work from Pruthi et al. [69] find the influential training examples by tracking the change in the test loss between different model checkpoints that are saved during the training time.

### 2.2.3 Explanations by saliency maps

The motivation behind saliency maps is to highlight those features in the input, which are essential to be classified as a certain class. Simonyan et al. [78] and Zeiler and Fergus [101] present two very similar ideas to create saliency maps for deep neural networks. Simonyan et al. [78] use the gradient of the class score with respect to the image, while Zeiler and Fergus [101] use *DeconvNet*, a way to backtrack the class scores through the network. Both approaches are very similar. The key difference between the two is handling the flow of the gradient through the non-differentiable layers like ReLU. Simonyan et al. [78], zero the gradient of the neurons with negative input when propagating through the ReLU. On the other hand, Zeiler and Fergus

[101], zero the gradient of the neurons with negative gradients. Both of the methods produce very similar results.

Springenberg et al. [80] extend the work of Zeiler and Fergus [101] to *Guided backpropagation*. They found that for ReLU, by zeroing the gradient of the neuron with either input or gradient having negative values creates more informative saliency maps. They achieve this by doing the element-wise multiplication of the gradient and the input at each layer. The main issue with these approaches arises from non-differential functions like ReLU. As the partial derivatives with respect to each feature are calculated, the gradient is inherently discontinuous over the entire input and produces artifacts [76, 82].

Bach et al. [5] create saliency maps by propagating relevance scores and distributing the score in proportion to the activation of previous layers. They call it *Layer-wise Relevance Propagation (LRP)*. Since it does not involve propagating the gradient with respect to the input, it also avoids the non-differentiability problem of layers like ReLU. *DeepLift* [76] gets around the problem of this non-differentiability by using discrete gradients. Instead of backpropagating the score of the given image through the network, they propagate the difference in the scores with respect to the input.

Sundararajan et al. [82] suggest that DeepLift and LRP suffer from *Implementation Invariance*: two networks are functionally equivalent if their outputs are equal for all inputs, despite having very different implementations. Methods like LRP and DeepLift replace gradients with discrete gradients and use a modified form of backpropagation to compose discrete gradients into attributions. Unfortunately, the chain rule does not hold for discrete gradients in general. Therefore, these methods fail to satisfy implementation invariance. The authors propose an approach to overcome this issue called *Integraded Gradient*. In this approach, images are perturbed over a continuous domain from a baseline image (all zeroes) to the current image, and the sensitivity of each pixel with respect to prediction is integrated over the

spectrum to give an approximate attribution score for each pixel.

Zhou et al. [109] propose *Class Activation Map (CAM)*, where instead of trying to propagate back the gradients, authors use the activation maps of the final convolutional layer weighted with the class weights to create downsampled relevance map of the input pixels. The downsampled saliency map is then upsampled to create a heatmap for the input image. One big shortcoming of this approach is, it can only be applied to the fully convolutional networks. Moreover, it also requires global average max-pooling at the end. Selvaraju et al. [74] extend this idea to *GRAD-CAM*, which can be applied in any network. Unlike CAM, in GRAD-CAM, the activation map of the final convolution layer is weighted using the gradient information of the target class score. These gradients are calculated with respect to the features of the final convolution layer.

Fong and Vedaldi [27] propose an approach to refine the saliency mask, specifically to find the minimal mask to define the saliency. The idea is to learn a perturbed image that blurs a certain region of the original image to have a very low activation value for the given class. The blurred region acts as the saliency map. While these methods are a popular class of tools for interpretability, Adebayo et al. [1] suggest that relying on visual assessment is not adequate and can be misleading. Rudin [73] present examples where, for a given image, the saliency maps for multiple classes are the same.

## 2.3 Counterfactual explanations

For a given input, counterfactual explanations aim to answer the following question: what input features differentiate a model's decision between two classes. Formally, a counterfactual explanation seeks the minimal change to a given input that will change a classifier decision in a prescribed way. Given an input $\bar{\mathbf{x}}$ predicted as class $c$ by model $f$, the problem of finding counterfactual explanations can be

formulated as:

$$\arg\min_{\mathbf{x}} E(\mathbf{x}, \bar{\mathbf{x}}) \qquad \text{s.t.} \quad f(\mathbf{x}) = y \tag{2.2}$$

where, $E(\mathbf{x}, \bar{\mathbf{x}})$ is the cost of changing features of $\bar{\mathbf{x}}$, and $y$ is the target class. Counterfactual explanations can be seen as a form of knowledge extraction from a trained machine learning model. This is the traditional realm of data mining, particularly in business and marketing [2, 4, 39, 40, 45, 93]. However, the precise formulation of counterfactual explanations as optimization problems given a classifier, source instance and target class (eq. (2.2)) and the various works exploring this research topic are quite recent.

Wachter et al. [89] present one such formulation, where the class constraint is moved to objective function as a penalty. They illustrate their approach to the problem of law school admissions and risk factors likely to increase a patient's chance of developing diabetes. Their formulation only applies to differentiable models and real-valued features. Similar work by Martens and Provost [56], is explicitly designed to use on discrete data and focuses on a particular problem where words need to be removed from a website for it to be no longer be classified as objectionable content.

Ustun et al. [87] formulate the counterfactual problem as an integer program for linear models with binary classification. Their formulation can handle both continuous and discrete features separately but not together. Rudin [73] address this issue by formulating the problem as a mixed-integer program. Carreira-Perpiñán and Hada [13] propose an approach that takes model structure into consideration and solves the counterfactual problem for linear models with Newton's method by reformulating the Hessian. This results in solving the problem within milliseconds, even for very large dimensional problems. However, the formulation is applicable only to problems with real-valued features.

Yang et al. [96], motivated by customer relationship management, seek to infer actions from a binary classification tree (attrition vs no attrition), specifically to move a group of instances (customers) from some source leaves to some target leaves

of the tree. The problem is different from a standard counterfactual explanation and is restricted to categorical features only. It takes the form of a maximum coverage problem, which is NP-complete and is approximated with a greedy algorithm. Bella et al. [7] consider a restricted form of counterfactual explanation over a single, "negotiable" feature, which must be continuous and satisfy certain sensitivity and monotonicity conditions.

Cui et al. [19] formulate a type of counterfactual problem for binary classification forests, show it is NP-hard, and encode it as an integer linear program, which can be (approximately) solved by existing solvers. It is practical only for low-dimensional problems, and even then it takes seconds or minutes for one instance. Kanamori et al. [44] extend this formulation to generate counterfactuals to support $\ell_1$ distance as the cost function. They also used plausibility constraints grounded on the Local Outlier Factor (LOF) score, where the goal is to generate counterfactuals that are closer to the training data by adding distances to a set of training instances as a penalty to the cost function. More recently, Parmentier and Vidal [67] further improved the integer programming formulation by relaxing the binarization of the numerical features to achieve better runtime performances. However, since the formulation still requires to solves an integer program, similar to the other two, this approach is limited to small problems.

Some approaches like [52, 85] use heuristic approaches to solve the counterfactual problem for tree ensembles. Lucic et al. [52] use a gradient-based algorithm that approximates the splits of the decision trees through sigmoid functions. On the other hand, Tolomei et al. [85] propose an approximate algorithm based on propagating the source instance down each tree towards a leaf. However, these approaches are slow for large forests and, most of the time, fail to find a feasible solution as the problem size grows bigger in terms of features and the number of trees.

Although finding a counterfactual explanation for a forest cannot be solved exactly, for a single tree, it can be done exactly and efficiently [11, 12, 36]. Carreira-

Perpiñán and Hada [11] achieve this by solving the problem in each target leaf separately and picking the solution with the lowest cost. Hada and Carreira-Perpiñán [36] extends this formulation to generate counterfactual explanations for regression trees.

Recently, some model-agnostic approaches [33, 46, 92] have been proposed, which treats classifier as a black-box to generate counterfactual explanation. Karimi et al. [46] generate a counterfactual explanation by creating a logical rule for a given model. Then they find an input that can satisfy those logical rules using SAT solvers. This approach cannot be used for complex models, as expressing a complex model in terms of logical rules is difficult. Guidotti et al. [33] use a genetic algorithm to generate neighbors around the input and then uses decision trees to locally approximate the model. Then, the counterfactual explanation is created based on the minimum number of splits in the decision tree. However, it generates the counterfactual, which might not be closest to the input. In practice, these kinds of model-agnostic approaches do not scale beyond small dimensional problems and often suffer from the problem of being a bad mimic of the original model.

Van Looveren and Klaise [88] propose a model-agnostic approach focused on continuous data, especially images. Because counterfactual explanations for images are usually indistinguishable from the input, they propose to add the class prototypes as a penalty term in the objective function. The class prototype represents the mean of training data features (calculated by an autoencoder) corresponding to the target class. This helps the counterfactual explanation to have a visible difference from the original image. What-If Tool [91] restrict the instance search space to a finite set of instances (such as the training set of the classifier), so the optimization to find a counterfactual explanation involves a simple brute-force search, as in a database. While model-agnostic approaches are very general, they are computationally slow, particularly with high-dimensional instances, and give a poor approximate solution.

# Chapter 3

# Sampling the "Inverse Set" of a Neuron

In this section, we propose an approach to characterize the region of input space that excites a given neuron to a certain level; we call this the *inverse set* [34, 35]. This inverse set is a complicated high-dimensional object that we explore by an optimization-based sampling approach. Inspection of samples of this set by a human can reveal regularities that help to understand the neuron. This goes beyond approaches that were limited to finding an image which maximally activates the neuron [78] or using Markov chain Monte Carlo to sample images [65]. Additionally, approaches like Nguyen et al. [65] are very slow, generate samples with little diversity, and lack control over the activation value of the generated samples.

## 3.1 The inverse set of a neuron: definition

We say an input $\mathbf{x}$ is in the inverse set of a given neuron having a real-valued activation function $f$, if it satisfies the following two properties:

$$z_1 \leq f(\mathbf{x}) \leq z_2 \quad \text{and} \quad \mathbf{x} \text{ is a valid input} \tag{3.1}$$

where, $z_1$, $z_2 \in \mathbb{R}$ are activation values of the neuron. $\mathbf{x}$ being a valid input means the image features are in the valid range (say, pixel values in [0,1]) and it is a natural looking image.

For a simple model, the inverse set can be calculated analytically. For example, consider a linear model with logistic activation function $\sigma(\mathbf{w}^T\mathbf{x} + c)$, and all valid inputs to have pixel values between [0,1]. For $z_2 = 1$ (maximum activation value) and $0 < z_1 < z_2$, the inverse set will be the intersection of the half space $\mathbf{w}^T\mathbf{x}+c \geq \sigma^{-1}(z_1)$ and the [0,1] hypercube.

A simple way to do this is to select all the images in the training set that satisfy eq. (3.1), but this may rule out all images. A neuron may "like" certain aspects of a training image without being sufficiently activated by it, or, in other words, the images that activate a given neuron need not look like any specific training image. Therefore, we need an efficient algorithm to sample the inverse set.

## 3.2 Sampling the inverse set of a neuron: an optimization approach

To generate the maximum activation image, the problem can be mathematically formulated as:

$$\arg\max_{\mathbf{x}} f(\mathbf{x}) + \mathcal{R}(\mathbf{x}) \tag{3.2}$$

where real-valued function $f$ gives the activation value of the given neuron for an input image $\mathbf{x}$. $\mathcal{R}$ is a regularizer which makes sure that the generated image $\mathbf{x}$ looks like a real image. This is the same objective function used in Simonyan et al. [78], Yosinski et al. [99], Nguyen et al. [64], and others; where, $\mathcal{R}$ is replaced by their hand-crafted regularizers. As mentioned in Nguyen et al. [64], it mostly produces the same images for a given neuron. However, to construct the sample $S = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$

that covers the inverse set, generated images should be different from each other. So, we propose the following formulation to construct $S$ of size $n$ as a constraint optimization problem:

$$\underset{\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n}{\arg\max} \sum_{i,j=1}^{n} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \quad \text{s.t.} \quad z_1 \leq f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n) \leq z_2. \tag{3.3}$$

The objective function makes sure that the samples are different from each other and also satisfy eq. (3.1). However, this generates noisy-looking samples. To make them realistic we use an image generator network $\mathbf{G}$, which has been empirically shown to produce realistic images [25] when a feature vector $\mathbf{c}$ is passed as an input. Then we get:

$$\underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_n}{\arg\max} \sum_{i,j=1}^{n} \|\mathbf{G}(\mathbf{c}_i) - \mathbf{G}(\mathbf{c}_j)\|_2^2 \quad \text{s.t. } z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_n)) \leq z_2. \tag{3.4}$$

We observe that using Euclidean distances directly on the generated images $\mathbf{G}(\mathbf{c})$ is very sensitive to small changes in their pixels. Instead, we compute distances on a low-dimensional encoding $\mathbf{E}(\mathbf{G}(\mathbf{c}))$ of the generated images, where $\mathbf{E}$ is obtained from the first few layers of a deep neural network trained for classification. Then we have our final formulation of the optimization problem over the $n$ samples $\mathbf{G}(\mathbf{c}_1), \ldots, \mathbf{G}(\mathbf{c}_n)$:

$$\begin{aligned} \underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_n}{\arg\max} & \sum_{i,j=1}^{n} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 \\ & \text{s.t.} \quad z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_n)) \leq z_2. \end{aligned} \tag{3.5}$$

Now to generate the samples, initialize $\mathbf{c}$ with random values and then optimize eq. (3.5) using augmented Lagrangian [66].

In theory, eq. (3.5) is enough to generate the sample $S$. But in practice, as the $n$ gets larger which is required to correctly sample the inverse set, eq. (3.5) pose two issues: First, because of the quadratic complexity of the objective function over

the number of samples $n$, it is computationally expensive to generate many samples. Second, since it involves optimizing all codes together, for larger $n$, it is not possible to fit all in the GPU memory. In the next section, we describe a much faster and less computationally expensive approach to create the inverse set.

## 3.3 Sampling in feasible region

In this section, we solve the problem for sampling the set $S$ described in eq. (3.5) with a faster approach. For this, we apply two approximations.

First, we solve the problem in an inexact but good enough way. The sum-of-all-pairs objective is not a strict necessity; it is really a mechanism to ensure the diversity of the samples and coverage of the inverse set. We observe that this is already achieved by stopping the optimization algorithm once the samples enter the feasible set, by which time they already are sufficiently separated.

Second, we create the samples incrementally, $K$ samples at a time (with $K \ll n$). For the first $K$ samples (which we call *seeds* ($\mathbf{C}_0$)) we optimize eq. (3.5), initializing the code vectors ¸with random values and stopping as soon as all $K$ samples are in the feasible region. These samples are then fixed. The next $K$ samples are generated by the following equation:

$$\begin{aligned} \underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_K}{\arg\max} & \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 + \sum_{i=1}^{K}\sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_y))\|_2^2 \\ \text{s.t.} \quad & z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_K)) \leq z_2 \quad \text{and} \quad \mathbf{c}_y \in \mathbf{C}_0. \end{aligned} \tag{3.6}$$

The first part of the equation "$\sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2$," is similar to that of eq. (3.5), means samples should be apart from each other. On the other hand, the second part of the equation "$\sum_{i=1}^{K}\sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_y))\|_2^2$," makes sure that the generated samples should be far apart from the previous ones. The presence of


Figure 3.1: Samples generated using our sampling approach for the neuron number 981 in the fc8 layer of the CaffeNet [43], which represents **Volcano** class. Each row contains 10 samples picked from 500 samples generated with the activation range mentioned on the left side. Generated samples are not just photo-realistic but also very diverse in nature, characterizing input space around certain activation very well. Unlike previous approaches [64, 65]; generated samples not only contain volcanoes but also lava flowing through water and on land and the ash cloud after the volcanic eruption. These kinds of samples have never been seen before. The first row contains samples of high activation, and the last row, which has samples with low activation, both have volcanoes, but the presence of lava and smoke creates a huge difference in the activation value of the neuron. Giving us a clear understanding how the amount of lava and smoke impact the activation of the neuron, higher the amount of lava and smoke more the activation.

constraints makes sure that generated samples stay in the feasible region.

Now to pick next $K$ samples, we initialize them to the previous $K$ samples ($\mathbf{C}_0$) and take a single gradient step in the Augmented Lagrangian optimization

of eq. (3.6). This gives $K$ new samples ($\mathbf{G}(\mathbf{c}_i)$) which we fix, and the process is repeated until we generate the desired $n$ samples.

### 3.3.1 Intersection of inverse sets

Our method also allows us to visualize the intersection of multiple inverse sets. This can be achieved by modifying the constraints in eq. (3.5) as:

$$\begin{aligned} &\underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_n}{\arg\max} \sum_{i,j=1}^{n} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 \\ \text{s.t.} \quad & z_1^{(1)} \leq f_1(\mathbf{G}(\mathbf{c}_1)), \ldots, f_1(\mathbf{G}(\mathbf{c}_n)) \leq z_2^{(1)}, \\ & \vdots \\ & z_1^{(p)} \leq f_p(\mathbf{G}(\mathbf{c}_1)), \ldots, f_p(\mathbf{G}(\mathbf{c}_n)) \leq z_2^{(p)} \end{aligned} \tag{3.7}$$

and in eq. (3.6) as follows:

$$\begin{aligned} &\underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_K}{\arg\max} \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 + \sum_{i=1}^{K} \sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_y))\|_2^2 \\ \text{s.t.} \quad & z_1^{(1)} \leq f_1(\mathbf{G}(\mathbf{c}_1)), \ldots, f_1(\mathbf{G}(\mathbf{c}_K)) \leq z_2^{(1)}, \\ & \vdots \\ & z_1^{(p)} \leq f_p(\mathbf{G}(\mathbf{c}_1)), \ldots, f_p(\mathbf{G}(\mathbf{c}_K)) \leq z_2^{(p)} \text{and} \quad \mathbf{c}_y \in \mathbf{C}_0 \end{aligned} \tag{3.8}$$

where, $f_1, \cdots, f_p$ are real valued activation functions for different neurons and $(z_1^{(1)}, z_2^{(1)}), \cdots, (z_1^{(p)}, z_2^{(p)})$ are corresponding activation values.

Figure 3.5 shows samples from the intersection of two different inverse sets, corresponding to neurons in the same layer. Eq. (3.8) can also be used to generate samples from inverse set intersection of neurons in different layers, as shown in figure 3.6. Generated samples have a face in the middle of the images. When these samples passed through the network, they excite both neurons in their corresponding activation range.

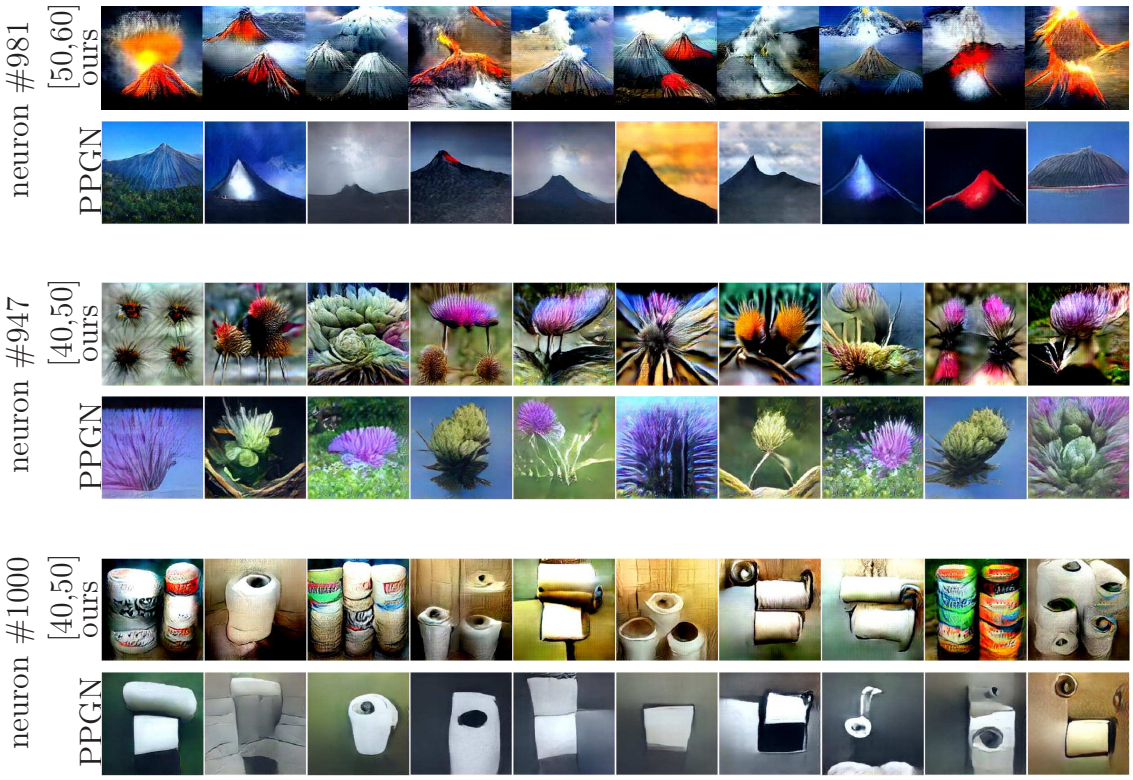


Figure 3.2: The first, third, and fifth rows contain 10 samples picked from 500 samples generated by our sampling approach to cover the inverse set for the neuron number 981 (represents **Volcano** class), 947 (represents **Cardoon** class), and 1000 (represents **Toilet paper** class) respectively. All three neurons are from layer fc8 of CaffeNet [43]. For the first row (**Volcano** class), the activation range is [50,60]; for the third (**Cardoon** class) and the fifth row (**Toilet paper** class), the range is [40,50]. The second, fourth, and sixth rows show samples generated for the same neurons by the sampling approach from Nguyen et al. [65]. The activation range for the samples from Nguyen et al. [65] is not guaranteed to be in any fixed range like ours.

## 3.4 Experiments

In all our experiments, $f$ is a pre-trained CaffeNet [43], which had been trained on the ImageNet dataset [21]. Using previous papers' [63, 65] naming convention, we will also call last three fully connected layers in CaffeNet [43] as fc6, fc7, and fc8.

| **Neuron in fc8** | Ours | PPGN |
|---|---|---|
| 981(volcano class) | 5.58 | 4.82 |
| 947(cardoon class) | 6.00 | 4.99 |
| 1000(toilet paper class) | 5.83 | 4.79 |

Table 3.1: Comparison between the proposed method and PPGN; in terms of mean pairwise Euclidean distance between the codes ($\mathbf{E}(\mathbf{G}(\mathbf{c}_i))$) of the generated samples (all 500 samples in figure 3.2). Higher values mean generated samples are more diverse.

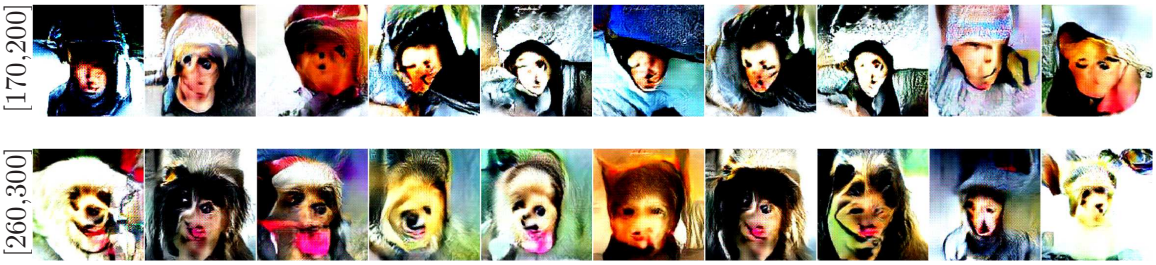


Figure 3.3: Both rows contain 10 samples out of 500 samples which are generated to cover the inverse set for neuron number 197 in the conv5 layer of CaffeNet [43]. This neuron is known to detect faces [99]. In the first row, the activation range is $[170, 200]$, and samples are mostly human faces. On the other hand, the second row, which has higher activation range ($[260, 300]$), contains dog faces with fur. This shows the neuron prefers faces with fur compared to clean faces.

fc8 is the last layer before softmax. In all our experiments, $\mathbf{E}$ is pre-trained CaffeNet [43]. However, the network has been shortened to the fc6 layer, which is the first fully connected layer. The output of $\mathbf{E}$ is a vector of size 4096. $\mathbf{G}$ is a pre-trained generative network from Nguyen et al. [65] [1].

Unlike previous visualization approaches [63, 64, 65] (second row of figure 3.2), the generated samples are far more diverse and rich in information. Our approach allows us to look at the samples which excite the neuron at different activation levels; this was not possible before. In doing so, it uncovers samples that have never been

[1] `https://github.com/Evolving-AI-Lab/ppgn`


Figure 3.4: Figure shows the results of our sampling process over a different network. The network is a variant of AlexNet trained on MIT Places dataset [107]. The first row contains samples from the inverse set for neuron number 88 (**Gas station** class ), and the second row contains samples from the inverse set for neuron number 141 (**Phone booth** class). Both neurons are from the last layer before softmax, and the activation range is $[40, 50]$ in both cases.


Figure 3.5: First two rows show the samples generated by our sampling method for neuron number 664 (represents **Monastery**) and 862 (represents **Toilet seat**) respectively. Both neurons are from layer fc8 of CaffeNet [43]. The last row contains the samples from their intersection set. All three rows have an activation range of [40,50].

seen before. For instance, images that do not even contain a volcano instead contain lava flowing through water but still excite the neuron (figure 3.1).

Figure 3.2 shows some of the samples generated by our algorithm for a certain

Figure 3.6: Inverse set intersection of neurons from different layers of the network. The hidden neuron in both rows is number 197 (known to detect faces) from the conv5 layer, and the activation range is [260,300]. *Top row:* The other neuron is from the fc8 layer (neuron number 1000, **Toilet paper** class), with the activation range of [20,50]. *Bottom row:* Again the other neuron is from fc8 layer (neuron number 862, **Toilet seat** class), with the activation range of [40,50].

activation range with some other neurons and their comparison with Nguyen et al. [65]. Second row shows the generated samples for neuron number 947 which represents "Cardoon" class for $z_2 = 50$ and $z_1 = 40$. The generated samples not only have a more diverse color distribution compared to Nguyen et al. [65] (fourth row in the figure 3.2) but also show samples of different shapes and sizes that can only be seen in real-life images.

To further test whether our sampling truly generates diverse samples or not, we pick a rather difficult class – "Toilet paper", neuron number 1000. The reason for the difficulty is as the realistic-looking samples cannot just differ by having a different color like in the case of "Cardoon" class; they all should have the white color. The sampling method should pick the samples which are of different shapes or quantities. The fifth row in the figure 3.2 shows our results. The generated samples are very diverse; they not only show just a toilet roll or toilet rolls hung from a hanger but also show packed toilet rolls. In fact, samples contain packages with different shapes, labels, and even quantities. The first and ninth sample in the fifth row of figure 3.2 shows rolls packed in two packages but have altogether different packaging labels, while the third sample shows toilet rolls packed in 3 packages which also have

different packaging label than the other two. Our sampling method was even able to pick samples such that the number of toilet rolls is different, as shown in the second, fourth, and tenth samples. These types of samples can only be seen in real-life images, thus showing how perfectly our sampling method covers the inverse set. On the other hand, samples generated by using PPGN [65] (sixth row of figure 3.2) mostly contain rolls that are standalone or attached to a holder, clearly not diverse. This diversity also translates numerically, as shown in table 3.1.

We also tested our method for hidden neurons. We apply our algorithm for neuron number 197 in the conv5 layer (last convolution layer in the CaffeNet [43]). This neuron is known to detect faces [99]. We first apply our approach with activation range $[200, 170]$, and most of the generated samples produce human faces, as shown in the first row of figure 3.3. But as we apply the same process with activation range $[300, 260]$, dog's faces started to appear in the samples, as shown in the second row of the figure 3.3. Hence shows, this neuron starts getting more activation towards the faces with more fur compared to that of clean faces.

## 3.5 Conclusion

Admittedly, the goal of understanding what a neuron in a deep neural network may be representing is not a well-defined problem. It may well be that a neuron does represent a specific concept, but one which is very difficult to grasp for a human; or that one should look at what a group of neurons may be representing. That said, for some neurons, their preferred response does correlate well with intuitive concepts or classes, such as the volcano or cardoon examples we give. Our approach is to characterize a neuron's preference by a diverse set of examples it likes, which is something that people sometimes do in order to explain a subjective concept to each other. Also, it may be possible to extract specific concepts from this set of examples using data analysis techniques.

# Chapter 4

# Interpreting Deep Neural Networks Using Sparse Oblique Trees

In this section, we propose an approach to provide a global explanation for a given deep net, using decision trees [37, 38]. As mentioned in section 2.2.1 this is by itself not a new idea. What is new is the specific, novel type of tree we use, and how we apply it to a given deep net. Traditional tree learning algorithms typically construct trees where each decision node thresholds a single input feature. Although such trees are considered among the most interpretable models, this is only true if the tree is relatively small. Unfortunately, such trees often produce too low accuracy, and are wholly inadequate for high-dimensional complex inputs such as pixels of an image or neural net features. We capitalize on a recently proposed *Tree Alternating Optimization (TAO)* algorithm [10, 14] which can learn far more accurate trees that remain small and very interpretable because each decision node operates on a small, learnable subset of features. Next, we apply the tree to an internal layer of the deep net, hence mimicking it's remaining (classifier) layers, rather than attempting

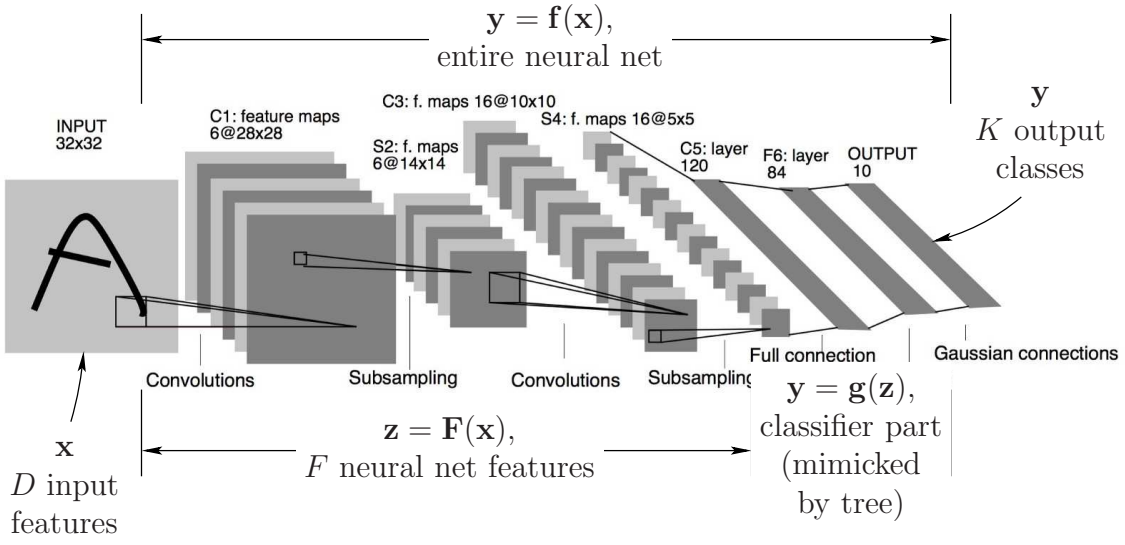


Figure 4.1: Mimicking part of a neural net with a decision tree. The figure shows the neural net $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{F}(\mathbf{x}))$, considered as the composition of a feature extraction part $\mathbf{z} = \mathbf{F}(\mathbf{x})$ and a classifier part $\mathbf{y} = \mathbf{g}(\mathbf{z})$. For example, for the LeNet5 neural net of [51] in the diagram, this corresponds to the first 4 layers (convolutional and subsampling) followed by the last 2, fully-connected layers, respectively. The "neural net feature" vector $\mathbf{z}$ consists of the activations (outputs) of $F$ neurons, and can be considered as features extracted by the neural net from the original features $\mathbf{x}$ (pixel values, for LeNet5). We use a sparse oblique tree to mimic the classifier part $\mathbf{y} = \mathbf{g}(\mathbf{z})$, by training the tree using as input the neural net features $\mathbf{z}$ and as output the corresponding ground-truth labels.

to mimic the entire deep net. This allows us to probe the relation between deep net features and classes. As a subproduct, inspection of the tree allows us to construct a new kind of adversarial attacks where we manipulate the deep net features via a mask to block a specific set of neurons. This gives us surprising control on what class the deep net will output. Among other possibilities, we can make it output the same, desired class for all dataset instances; or make it never output a given class; or make it misclassify certain pairs of classes. Our approach also allows us to study the relation between deep net features (neurons' activation) and output classes.

## 4.1 Mimicking the network using sparse oblique decision trees

Our overall approach is as follows (see figure 4.1). Assume we have a trained deep net classifier $\mathbf{y} = \mathbf{f}(\mathbf{x})$, where input $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^K$. We can write $\mathbf{f}$ as: $\mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{F}(\mathbf{x}))$, where $\mathbf{F}$ represents the features-extraction part ($\mathbf{z} = \mathbf{F}(\mathbf{x}) \in \mathbb{R}^F$), and $\mathbf{g}$ represents the classifier part ($\mathbf{y} = \mathbf{g}(\mathbf{z})$). Then:

1. Train a sparse oblique tree $y = T(\mathbf{z})$ with TAO on the training set $\{(\mathbf{F}(\mathbf{x}_n), y_n)\}_{n=1}^N \subset \mathbb{R}^F \times \{1, \dots, K\}$. Choose the sparsity hyperparameter $\lambda \in [0, \infty)$ such that $T$ has close to the highest validation accuracy and is as sparse as possible.

2. Inspect the tree to find interesting patterns about the deep net.

Our goal is to achieve a tree that both mimics well the deep net and is as simple as possible.

Step 2 is purposely vague. There is probably a wealth of information in the tree regarding the features' meaning and effect on the classification, both at the level of a specific input instance and globally. Here, we focus on one specific pattern described next.

## 4.2 Manipulating the features of a deep net to alter its classification behavior

Our overall objective is to control the network prediction by manipulating the value of the deep net features $\mathbf{z} \in \mathbb{R}^F$. We do not alter the network weights, i.e., $\mathbf{F}$ and $\mathbf{g}$ remain the same. We just alter $\mathbf{z}$ into a masked $\overline{\mathbf{z}} = \boldsymbol{\mu}(\mathbf{z}) = \boldsymbol{\mu}^{\times} \odot \mathbf{z} + \boldsymbol{\mu}^{+}$ via a *multiplicative and an additive mask* $\boldsymbol{\mu}^{\times}, \boldsymbol{\mu}^{+} \in \mathbb{R}^F$, respectively (where "$\odot$" means

elementwise multiplication).

$$\text{Original net: } \mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{F}(\mathbf{x})) \tag{4.1}$$

$$\text{Original features: } \mathbf{z} = \mathbf{F}(\mathbf{x}) \tag{4.2}$$

$$\text{Masked net: } \overline{\mathbf{y}} = \overline{\mathbf{f}}(\mathbf{x}) = \mathbf{g}(\boldsymbol{\mu}(\mathbf{F}(\mathbf{x}))) \tag{4.3}$$

$$\text{Masked features: } \overline{\mathbf{z}} = \boldsymbol{\mu}(\mathbf{F}(\mathbf{x})) = \boldsymbol{\mu}(\mathbf{z}) \tag{4.4}$$

We construct $\boldsymbol{\mu}$ by inspecting the tree, specifically by observing the weights of each decision node of the tree. Intuitively, $\boldsymbol{\mu}$ represents the path from root to the target leaf. We demonstrate the detailed masking operation in figure 4.2.

### Diverting all instances to one child

We define the decision rule at a decision node $i$ as: "if $\mathbf{w}_i^T\mathbf{z} + b_i \geq 0$ then go to right child, else go to left child", where $\mathbf{w}_i \in \mathbb{R}^F$ is the weight vector and $b_i \in \mathbb{R}$ is the bias [1] . We also describe a mask for node $i$, that diverts all instances to one child– we call it NODE-MASK $= \{\boldsymbol{\mu}^\times, \boldsymbol{\mu}^+\}$.

NODE-MASK works as follows. Write $\mathbf{w}$ and $\mathbf{z}$ as $\mathbf{w} = (\mathbf{w}_0\ \mathbf{w}_-\ \mathbf{w}_+)$ and $\mathbf{z} = (\mathbf{z}_0\ \mathbf{z}_-\ \mathbf{z}_+)$, where $\mathbf{w}_0 = \mathbf{0}$, $\mathbf{w}_- < \mathbf{0}$ and $\mathbf{w}_+ > \mathbf{0}$ contain the zero, negative and positive weights in $\mathbf{w}$, and $\mathbf{z} \geq \mathbf{0}$ [2] is arranged according to that. Call $\mathcal{S}_0$, $\mathcal{S}_-$ and $\mathcal{S}_+$ the corresponding sets of indices in $\mathbf{w}$. Then $\mathbf{w}^T\mathbf{z} + b = \mathbf{w}_-^T\mathbf{z}_- + \mathbf{w}_+^T\mathbf{z}_+$ with $\mathbf{w}_-^T\mathbf{z}_- \leq 0$ and $\mathbf{w}^T{}_+\mathbf{z}_+ \geq 0$. So if $\mathbf{z}_- = \mathbf{0}$ then $\mathbf{w}^T\mathbf{z} + b \geq 0$ and $\mathbf{z}$ would go to the right child and if $\mathbf{z}_+ = \mathbf{0}$ then $\mathbf{w}^T\mathbf{z} + b \leq 0$ and $\mathbf{z}$ would go to the left child. Hence, NODE-MASK defined as follows: to go left, $\boldsymbol{\mu}^\times \in \{0, 1\}^F$ is a binary vector containing ones at $\mathcal{S}_-$, zeros at $\mathcal{S}_+$ and $*$ (meaning any value) at $\mathcal{S}_0$; and $\boldsymbol{\mu}^+ \geq \mathbf{0}$ is a vector containing small positive values at $\mathcal{S}_-$ and zero elsewhere. To go right, exchange "$-$" and "$+$" in the procedure.

[1] we assume $|b_i| \ll \|\mathbf{w}_i\|$

[2] We assume the deep net features are nonnegative: $\mathbf{z} = \mathbf{F}(\mathbf{x}) \geq \mathbf{0}$. This is true for ReLUs, which are used in most deep nets at present.

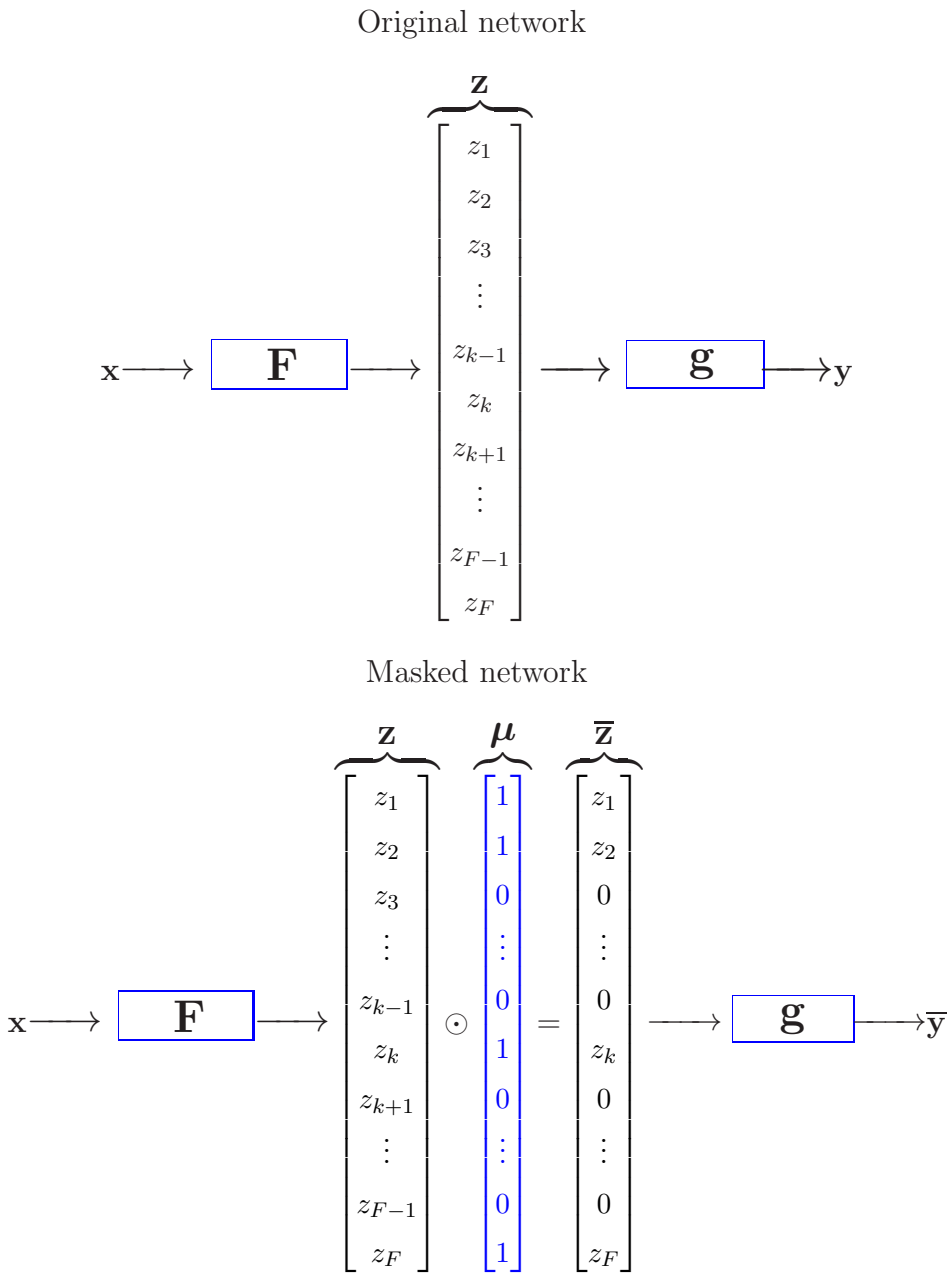


Figure 4.2: *Top:* original network. *Bottom:* masking operation in the network. The symbols' meaning is as follows: input $\mathbf{x}$, feature extraction part of the network $\mathbf{F}$, original features $\mathbf{z}$, binary mask created using the tree $\boldsymbol{\mu}$, modified features $\overline{\mathbf{z}}$, classifier part of the network $\mathbf{g}$, original output $\mathbf{y}$, and modified output $\overline{\mathbf{y}}$ .

**Masks**

Below we describe how to construct masks that change network prediction. For each case, we state the desired goal and the corresponding mask.

- ALL TO CLASS $k$: *let $k \in \{1, \ldots, K\}$. Classify all instances $\mathbf{x}$ as class $k$.*

  *Mask*: Apply NODE-MASK at each node $i$ in the path from root to class $k$. Also, in each NODE-MASK, set values at $\mathcal{S}_0$ to zero. To create the final multiplicative mask, take logical AND of all the multiplicative masks.

- ALL CLASS $k_1$ TO CLASS $k_2$: *let $k_1 \neq k_2 \in \{1, \ldots, K\}$. For any instance originally classified as $k_1$, classify it as $k_2$. For any other instance, do not alter its classification.* This case only works if the classes $k_1$ and $k_2$ are leaf siblings

  *Mask*: Simply apply NODE-MASK to the parent of the leaves of $k_1$ and $k_2$.

- NONE TO CLASS $k$: *let $k \in \{1, \ldots, K\}$. For any instance originally classified as $k$, classify it as any other class. For any other instance, do not alter its classification.*

  *Mask*: Apply ALL CLASS $k_1$ TO CLASS $k_2$ mask to each leaf with label $k$, and $k_2$ is $k$. At the end take logical AND of all the multiplicative masks.

## 4.3 Experiments

We evaluate our approach thoroughly on two deep nets.

- VGG16 [77] trained on a subset of 16 classes from ImageNet [21]. For this network we use $F = 8\,192$ neurons from its last convolutional layer to train the tree.

- LeNet5 trained on MNIST 10 digit classes [51], for which we select the $F = 800$ neurons at layer conv2 as features.

| Label id | Class |
|---|---|
| 0 | goldfish |
| 1 | bald eagle |
| 2 | goose |
| 3 | killer whale |
| 4 | Siberian husky |
| 5 | white wolf |
| 6 | tiger cat |
| 7 | lion |
| 8 | airliner |
| 9 | container ship |
| A | fire engine |
| B | school bus |
| C | speedboat |
| D | sports car |
| E | warplane |
| F | coral reef |

Table 4.1: Classes in our ImageNet subset and their id (for reference in other figures).

### 4.3.1 Results on VGG16

In this section, we show our results on VGG16 [77]. We trained this network over a subset of 16 classes of ImageNet [21] listed in table 4.1, and used $F = 8\,192$ neurons from its last convolutional layer to train the tree. Our VGG16 net achieves an error of 0.2% (training) and 6.79% (test). To train the tree, we start with an initial tree that is deep enough and a complete binary tree with random parameters. Next, we run TAO for a range of increasing $\lambda$ values (see figure 4.3). From there, we pick a tree with accuracy close to that of the deep net but as sparse as possible, which we will use as the mimic (figure 4.4). This tree ($\lambda = 1$) has an error of 0% (training) and 7.90% (test); it has 39 nodes and uses just 1 366 features (17% of the total 8 192). We also discuss a tree of somewhat lower accuracy but which has exactly one leaf per class (figure 4.5). This tree ($\lambda = 33$) has an error of 1.79% (training) and 9.56% (test); it has 31 nodes and uses just 408 features (5% of the total 8 192).

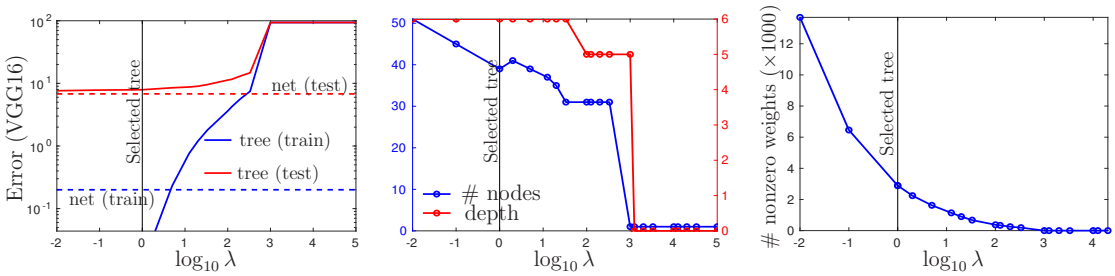


Figure 4.3: Classification error (training and test) and number of nodes and of nonzero weights of the trees as a function of $\lambda$ for VGG16. The vertical line indicates the tree we selected as the mimic ($\lambda = 1$).

### Inspecting the sparse oblique trees

Figure 4.4 shows the tree we use as the mimic. This tree has similar training and test error to those of VGG16, so we expect it to be a good mimic, which indeed happens (see figure 4.6). The top of the figure shows the class histogram at each node, i.e., the distribution of classes on the subset of training instances that a node receives. These histograms show how the tree hierarchically splits classes very crisply; indeed, it has only 20 leaves for 16 classes. At the bottom of the figure, the weight vector at each decision node is mostly sparse. This means only a very few features are used by the tree; indeed, 83% of the features are not used at any node. We test network performance by removing these features, and as shown in figure 4.6 (top row) network behaves the same as before. This suggests some of the features and hence neurons and weights of the net are practically redundant or perhaps code for properties that are useful for only a few specific instances.

Figure 4.5 shows a very interesting tree, obtained for a larger $\lambda$ value so that there is exactly one leaf per class. The weight vector at each decision node shows that very few nonzero weights are used at each node, yet the test error is reasonable. Also, its structure remains unchanged for a wide range of $\lambda$. So, nonzero features in this tree robustly classify most images. Inspecting it shows an intuitive hierarchy of classes that seem primarily related to the background or surroundings of the

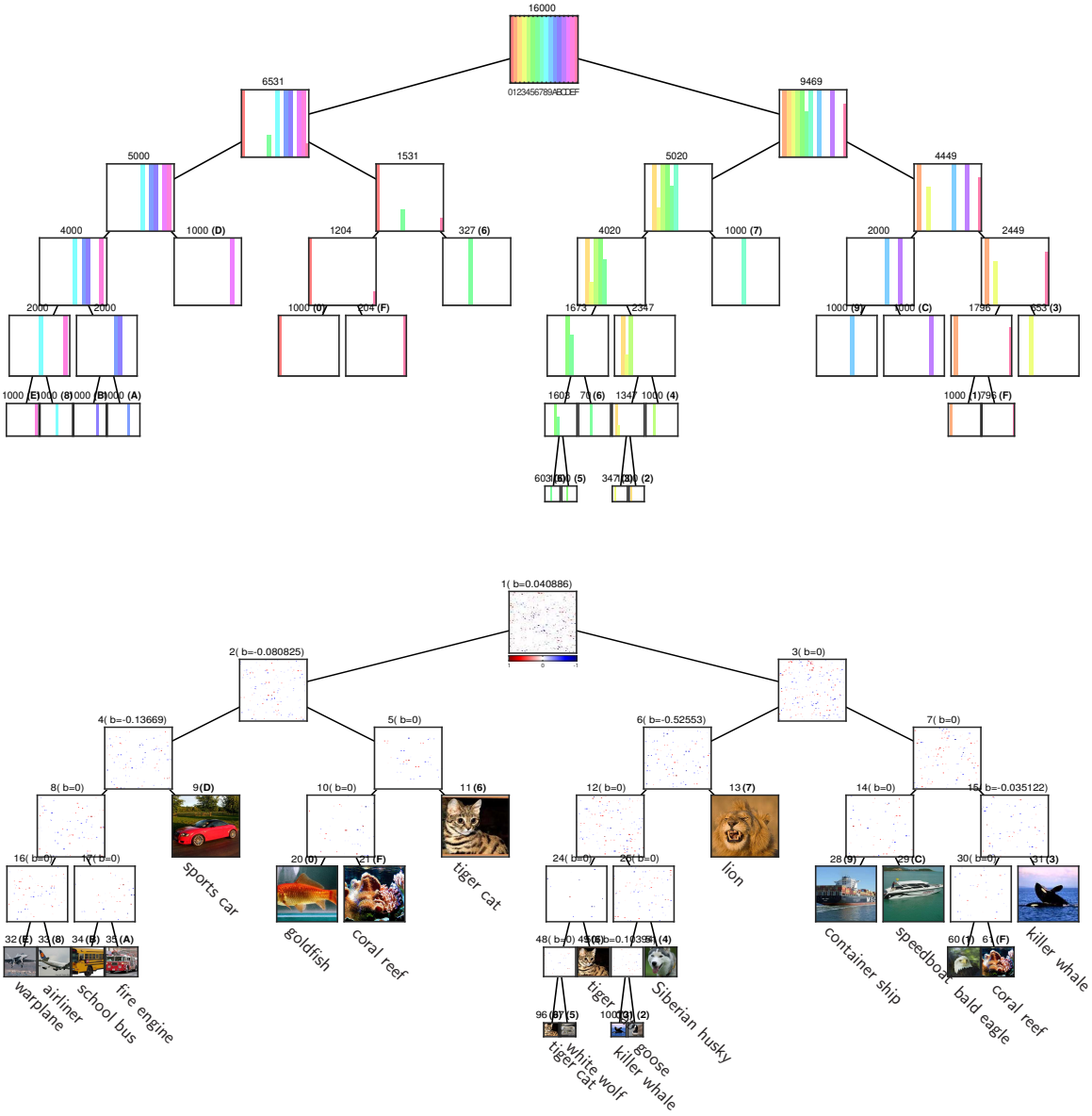


Figure 4.4: Tree selected as mimic for VGG16 features ($\lambda = 1$), with a training error of 0% and a test error of 7.90%. *Top*: class histograms; we show the number of training instances reaching the node and, for leaves, their label. *Bottom*: weight vector at each decision node and an image from their class at each leaf; we show the node index, bias (always zero) and, for leaves, their label. We plot the weight vector, of dimension 8 192, as a 91×91 square (the last pixels are unused), with features in the original order in VGG16 (which is determined during training and arbitrary, hence the random aspect of the images), and colored according to their sign and magnitude (positive, negative and zero values are blue, red and white, respectively). You may need to zoom in the plot.

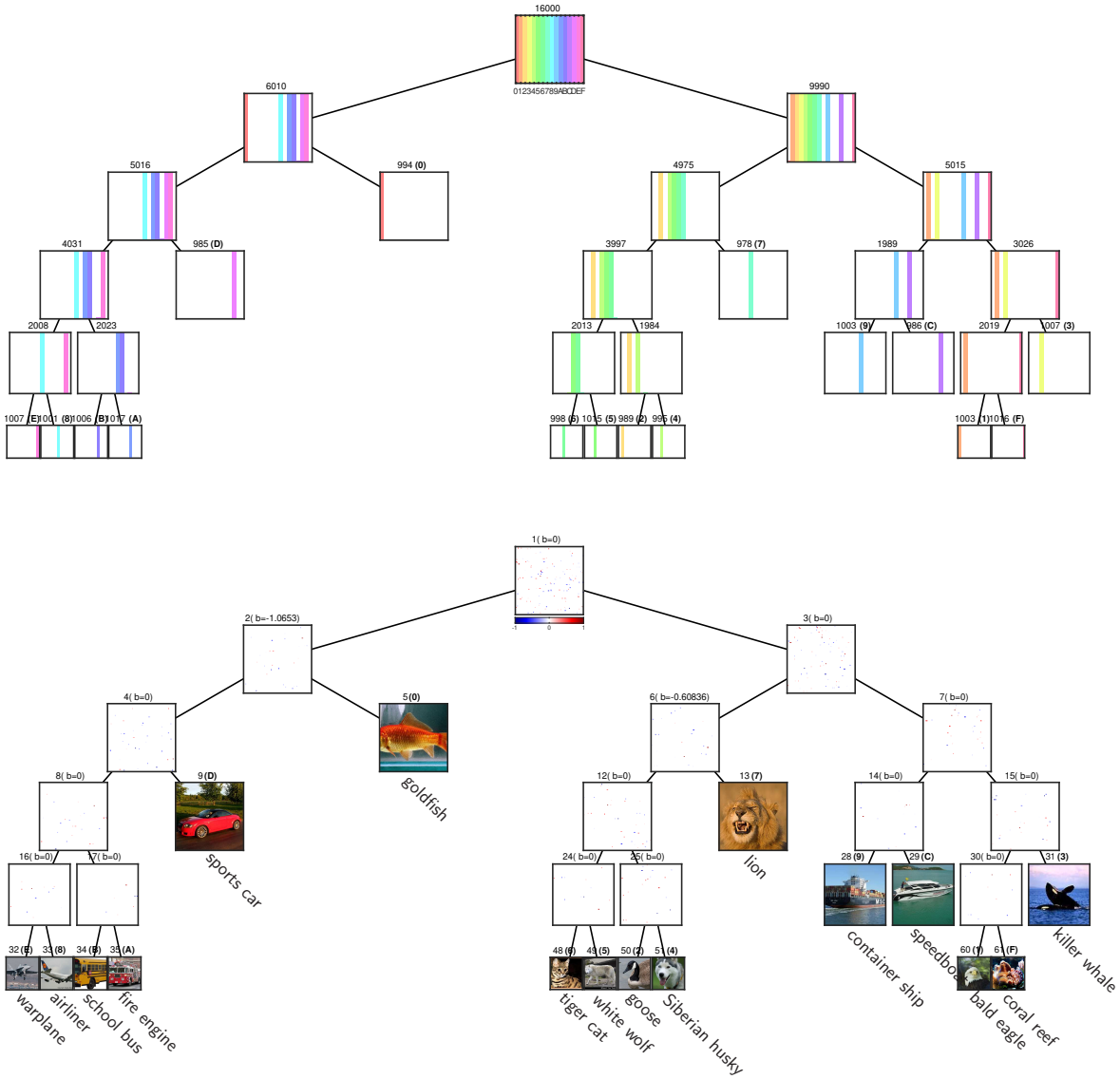


Figure 4.5: Like figure 4.4 but for $\lambda = 33$. This tree has exactly one leaf per class (total 16 classes), and a slightly higher error (training error 1.79%, test error 9.56%).

main object in the image. For instance, the leftmost tree consists of man-made objects often found on roads like: {warplane, airliner, school bus, fire engine and sports car}. However, the rightmost tree consists of both man-made objects {container ship, speedboat} and animals {killer whale, bald eagle, coral reef}, but both have the sea as common background. Similarly, the subtree in the middle contains animals {tiger cat, white wolf, goose, Siberian husky, lion} that are found mostly on the land environment. Yet {goldfish} appears in a single subtree quite separate from all other classes (not sea). On examining the training data, we observe that, in most of the training images {goldfish} is in fishbowls (not the sea). This is consistent with previous works that have found that, in some specific cases, the reason why a deep net classifies an object as a certain class is caused by the background or more generally by some confounding variables [71, 100].

### Manipulating the deep net features via masks

Figure 4.6–4.7 shows confusion matrices, which are self-explanatory, overall test and training instances, respectively. These confusion matrices are created by applying masks derived from the decision tree (figure 4.4). In general, all masks affect both the deep net and the tree in the same way. This is to be expected since the tree has a very similar error and confusion matrix as the net, but it is still surprising how well it works in most cases. This also suggests that every class is associated with certain group of neurons in the deep net.

### Illustration of the masks with an actual image

Figure 4.8 illustrates the mask behavior in an image not in the dataset. The middle column histograms show the deep net features (grouped by class) and how masking them drastically alters in a controlled way the softmax output (hence the class prediction; row 2 and 5). We also show how the mask correlates with superpixels

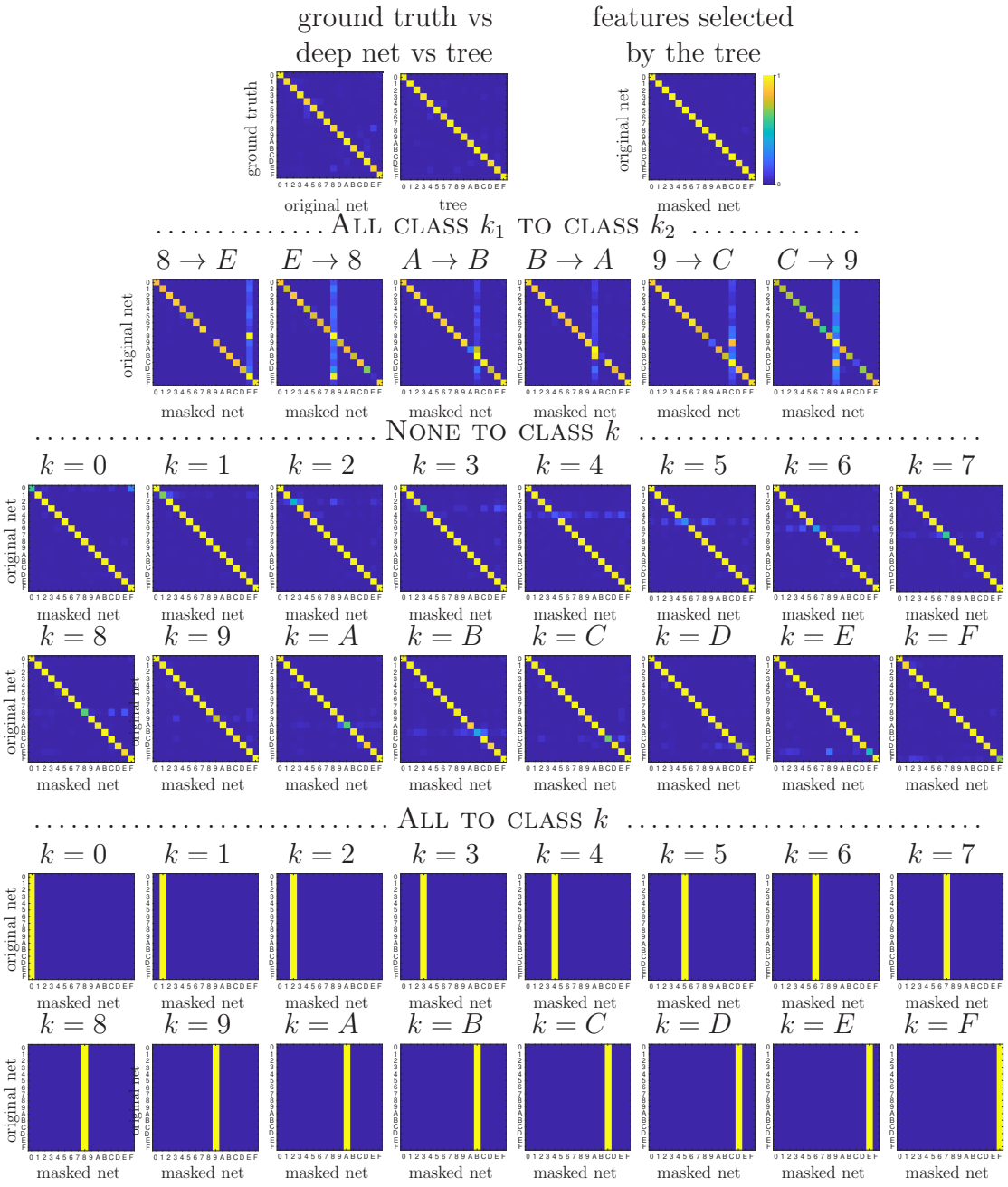


Figure 4.6: Confusion matrices for VGG16 (test set). *Top left*: ground-truth vs deep net, and deep net vs tree. *Top middle*: deep net vs deep net with only the features selected by the tree. *Top right*: ALL CLASS $k_1$ TO CLASS $k_2$ (selected examples). *Middle*: NONE TO CLASS $k$. *Bottom*: ALL TO CLASS $k$.

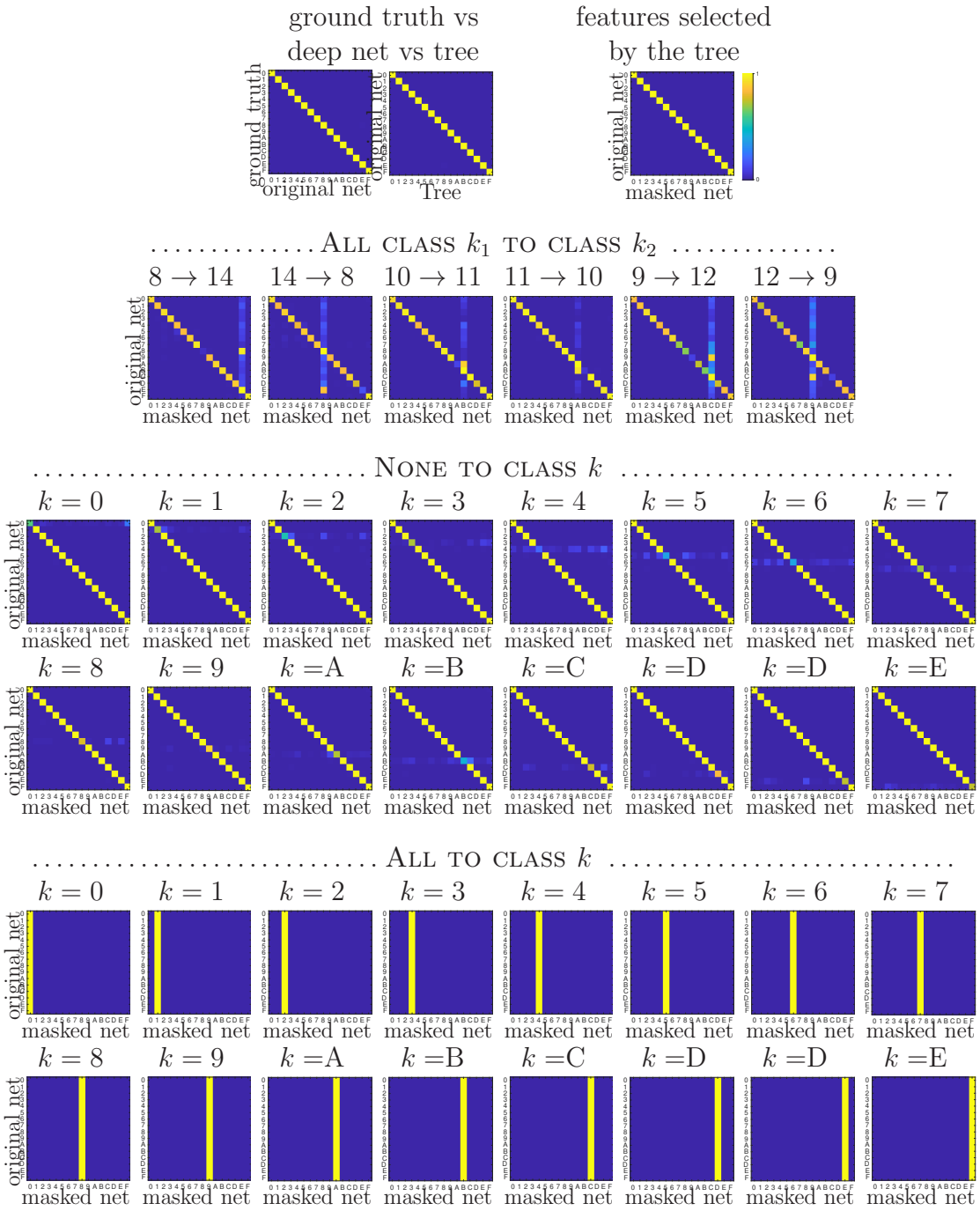


Figure 4.7: Like figure 4.6 but for the training set.

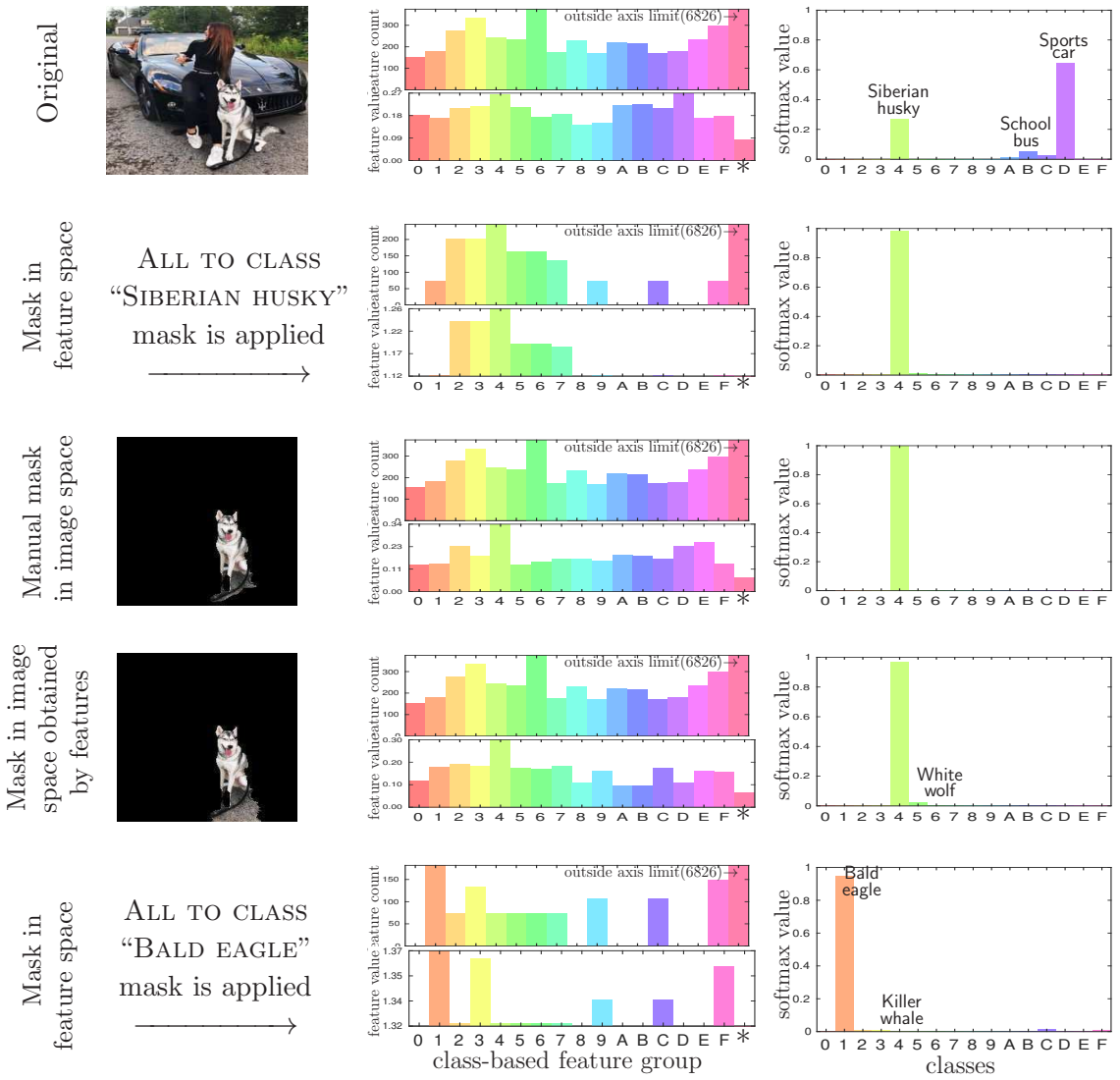


Figure 4.8: Illustration of masks for a particular image. Column 1 shows the image masks (when available). Column 2 summarizes the 8 192 feature values as two histograms: on the upper panel, the number of features in each class group (listed in the X axis as 0–F, where "*" means features not used by the tree); on the lower panels, the average feature value (neuron activation) per class group. Column 3 shows the histogram of corresponding softmax values. Row 1 shows the original image. Row 2 shows a mask in feature space to classify it as "Siberian husky". Row 3 shows a mask manually cropped in the image, whose features resemble those of row 2. Row 4 shows a mask in feature space obtained by finding the top-3 superpixels whose features most resemble those of the masked features of row 2. Row 5 shows a mask in feature space to classify the image as "bald eagle".

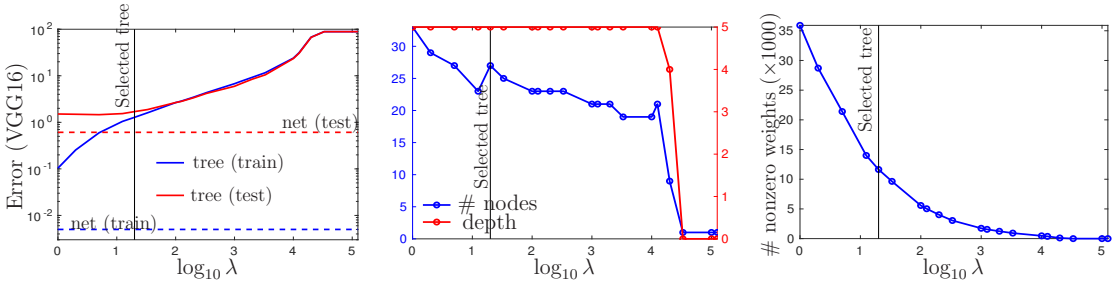


Figure 4.9: Classification error (training and test) and number of nodes and of nonzero weights of the trees as a function of $\lambda$ for LeNet5. The vertical line indicates the tree we selected as mimic ($\lambda = 20$).

in the image, either manually cropped (row 3) or optimized to invert the desired deep net features (row 4).

### 4.3.2 Results on LeNet5

We trained sparse oblique trees on features obtained by the LeNet5 neural net architecture for MNIST. The results are the same as VGG16: we are able to achieve a good mimic; the masks we construct work as desired in nearly all the instances; and the tree is highly interpretable.

We train a LeNet5 net on the 60 000 training images for MNIST, of 28×28 pixels. This network achieves a training and test error of 0.00545% and 0.61%, respectively. We select the F = 800 neurons at layer conv2 as features on which we train the tree. We used TAO with an initial tree structure of depth 5 (total 63 nodes) and random initial values for the weights at the nodes. Next, similar to VGG16 experiment we construct a collection of trees over a range of sparsity parameters $\lambda \in [0, \infty)$ (see figure 4.9). We selected as mimic the tree for $\lambda = 20$, which has depth 5 and only 27 nodes. It has an error of 1.28% (training) and 1.67% (test), which is very close to that of LeNet5, so we expect the tree to be a good mimic of the network.

Figure 4.10 shows the tree selected as mimic. The class histograms are similar to VGG16, showing tree hierarchically splits classes very crisply as it has 14 leaves for

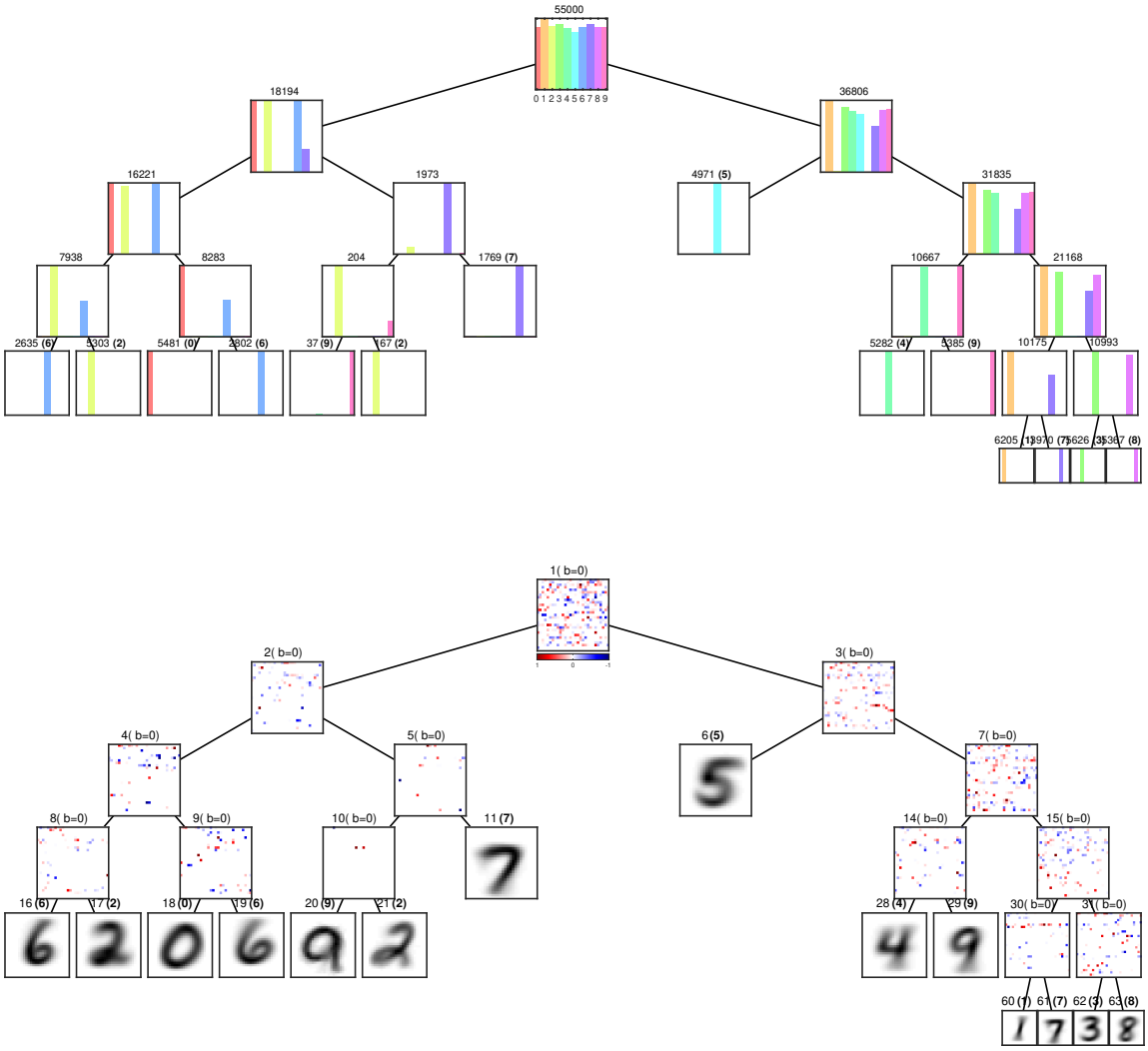


Figure 4.10: Tree selected as mimic for LeNet5 features ($\lambda = 20$). *Top*: class histograms; we show the number of training instances reaching the node and, for leaves, their label. *Bottom*: weight vector at each decision node and average of training instances at each leaf; we show the node index, bias (always zero) and, for leaves, their label. We plot the weight vector, of dimension 800, as a 29×29 square (the last pixels are unused), with features in the original order in LeNet5 (which is determined during training and arbitrary, hence the random aspect of the images), and colored according to their sign and magnitude (positive, negative and zero values are blue, red and white, respectively). You may need to zoom in the plot.

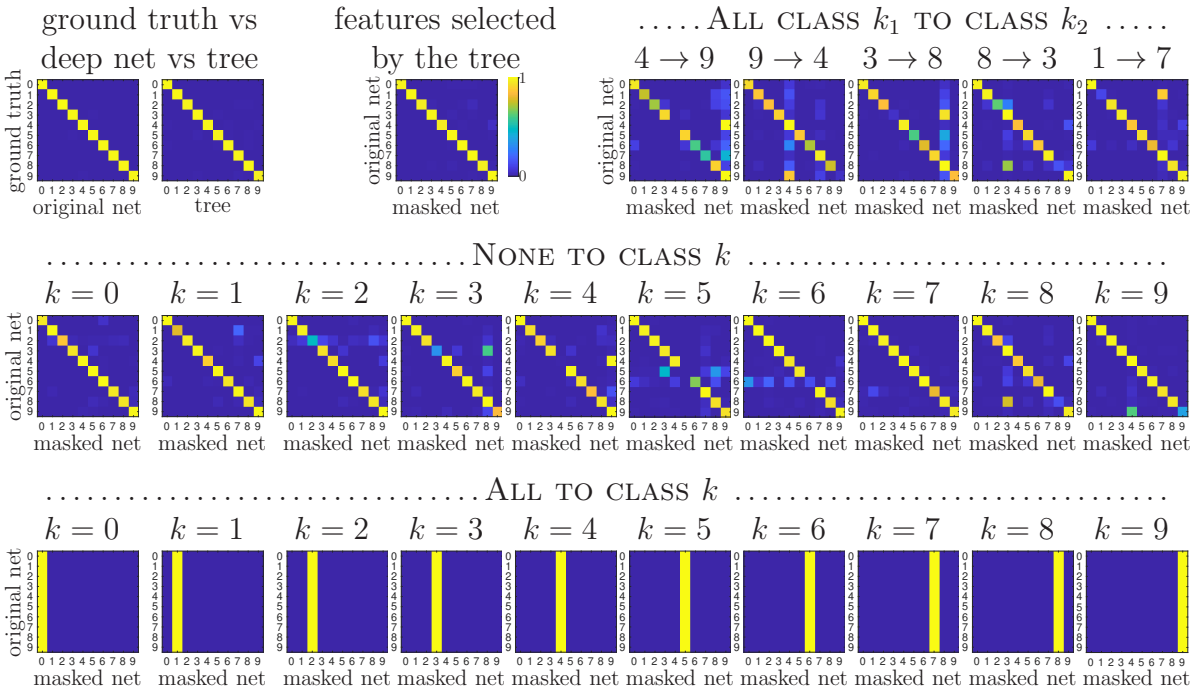


Figure 4.11: Confusion matrices for LeNet5 (test set). *Top left*: ground-truth vs deep net, and deep net vs tree. *Top middle*: deep net vs deep net with only the features selected by the tree. *Top right*: All class $k_1$ to class $k_2$ (selected examples). *Middle*: None to class $k$. *Bottom*: All to class $k$.

10 classes. The blurry average image at each leaf shows significant shape variability, indicating the features have successfully learned to ignore such within-class variability. Also, the weight vector at each decision node shows that very few features are used at each node; 295 features (37% of the total 800) are not used at any node, so their values are irrelevant for classification in the tree. This also holds nearly perfectly for the deep net (top rows in figures 4.11–4.12).

Figures 4.11–4.12 show the confusion matrices of our masks. Similar to VGG16 case, the masks work effectively in the entire dataset. The number of features we need to mask out in each case is very small, around 40 (out of 800 features), suggesting features learned by the network are special in that they seem to operate in very small groups associated with classes, rather than all features participating in each class.

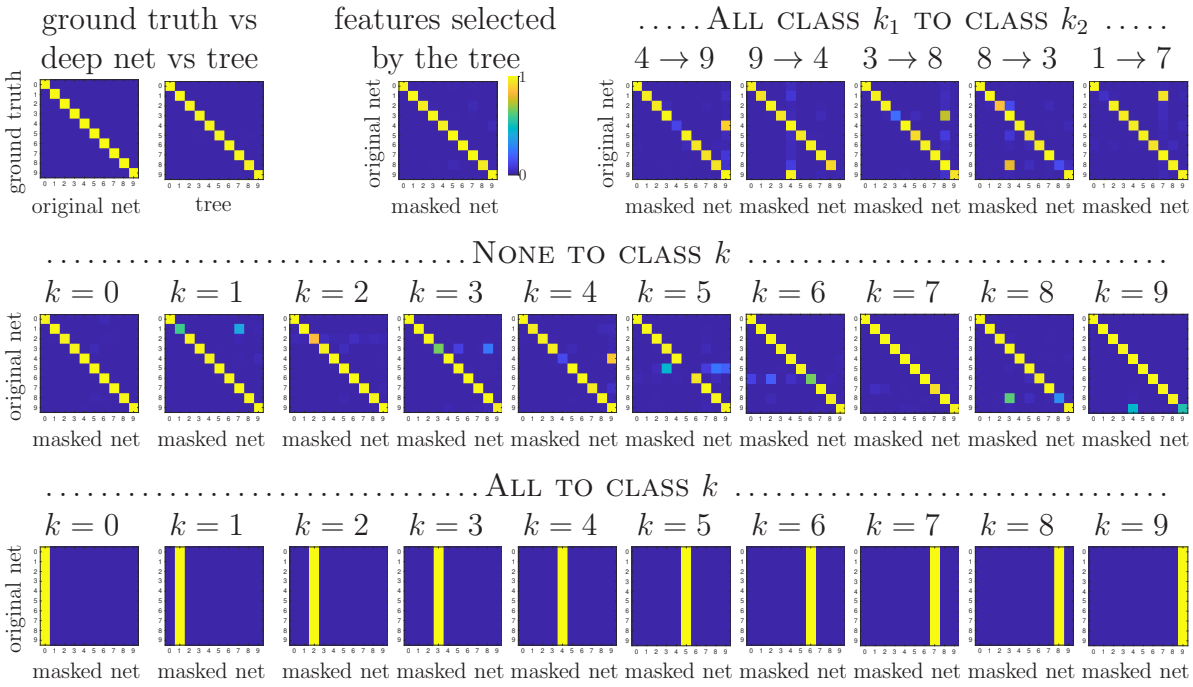


Figure 4.12: Like figure 4.11 but for the training set.

## 4.4 Conclusion

Decision trees are a good choice of interpretable classifier because they handle multiple classes naturally, perform feature selection automatically, have a hierarchical structure that promotes an increasing specialization from the root towards the leaves, and can be inspected. We demonstrate how sparse oblique trees can improve this by using a small number of features at each node, which explicitly shows the influence of groups of features on classes. This allows us to find a small subset of important features among thousands of possible features.

As shown in our experiments, deep net features participate in a coordinated way in predicting a class, where small groups of specific features encode information specific to that class. Any change in the activation of these class-specific neurons noticeably affects the deep network's ability to make decisions. This also aligns with

the hypotheses proposed in [50]. The authors hypothesize that the deep networks contain class-specific neurons, and if we remove these neurons, the deep network performance should be affected noticeably. However, it is surprising that the number of neurons in each group is very small compared to the possible number of neurons. This is probably partly due to the neural network being heavily overparameterized.

Our findings are remarkably similar to recent findings in visual neuroscience [16, 55] that show that very small groups of neurons (around 20) in mouse primary visual cortex seem to code for specific properties or behaviors. In fact, removing all visual input to the mouse and directly stimulating those neurons triggers the same behaviors—analogously to what our masks do.

Finally, our results are applicable to deep nets with specific weights. Since deep nets are typically overparameterized and have local optima, it is possible to obtain numerically very different weights depending on the initialization and optimization protocol. We have not explored how this may affect the resulting tree and the masks. We did observe that our results seem robust to the initialization of the tree itself.

# Chapter 5

# Interpretable Image Classification Using Sparse Oblique Decision Trees

In this section, we focus on another important aspect of interpretability, which concerns with understanding the data itself. Most explanation methods are restricted to provide information only about the model's prediction behavior [61, 71, 109] or what information is encoded by model's parameters [26, 34, 35, 63, 78], but provides very little insight about the training data on which the model is trained.

We can summarize small tabular datasets, but understanding the bigger datasets is difficult. For example, in image datasets, it is challenging to understand a given class's basic concept. Since there is so much irrelevant information, it is hard to understand what part of the image is important or what common concept defines a particular category of the class. Feature selection methods (Chi-Squared [28], Mutual Information [68], LASSO [41], and Kolmogorov-Smirnov statistic [22]) can help to understand these datasets. However, they leave a lot of questions unanswered. For example, in a classification task: we do not know how the classes are distributed in

the input space; is one class closer to another class, or how two classes or groups of classes differentiate from each other; or if any sub-groups exist in a given class, if yes what is the difference between them; or the selected features are optimal for a class, or sub-group of a class, or even an instance?

We can leverage our approach to interpret deep neural networks using sparse oblique trees to address the aforementioned issues. In the previous chapter, we used sparse oblique trees to understand what group of neurons are associated with a given class. However, it is unclear what these groups of neurons represent in the input space. We can explain the behavior of certain neurons individually (as described in chapter 3), but for a group of neurons, it is very difficult to do so. However, if we train the sparse oblique trees directly on the input features (pixels for image dataset), by extending our masking operation from the previous chapter, we can establish the relationship between classes and the input features.

Traditionally, axis-aligned trees are considered interpretable models, but these models are interpretable only for a small dataset. As the size of the dataset increases, the tree size grows by a large margin, making them very difficult to interpret (table 5.2 and figure 5.5). On the other hand, sparse oblique trees trained with TAO [10, 14] can achieve good accuracy while maintaining a smaller model size. This is due to the fact that, unlike axis-aligned trees that operate only on a single feature at each node, the sparse oblique tree operates on a small, learnable subset of features. Also, unlike traditional tree algorithms such as CART [9] or C4.5 [70], TAO monotonically decreases the objective function containing classification error and thus producing more accurate trees. It has been shown to outperform existing tree algorithms by a large margin [106], and to improve forests [15, 103, 104, 105].

## 5.1 Proposed approach

The overall approach is similar to the previous chapter.

1. Train a sparse oblique tree using TAO [10, 14] and pick the sparsity parameter such that the resultant tree is as sparse as possible but remains accurate enough.

2. Use the weights of decision nodes to extract relevant features from the dataset.

### 5.1.1 Interpreting weights of the decision nodes

For a given input $\mathbf{x} \in \mathbb{R}^d$ we define the decision rule at a decision node $i$ as follows: "if $\mathbf{w}_i^T\mathbf{x} + b_i \geq 0$, then go to right child, else go to the left child", where $\mathbf{w} \in \mathbb{R}^d$ is the weight vector of node $i$ and $b_i \in \mathbb{R}$ is the bias. Next, to extract features we apply following operation.

Write $\mathbf{w}$ and $\mathbf{x}$ as $\mathbf{w} = (\mathbf{w}_0\ \mathbf{w}_-\ \mathbf{w}_+)$ and $\mathbf{x} = (\mathbf{x}_0\ \mathbf{x}_-\ \mathbf{x}_+)$, where $\mathbf{w}_0 = \mathbf{0}$, $\mathbf{w}_- < \mathbf{0}$ and $\mathbf{w}_+ > \mathbf{0}$ contain the zero, negative and positive weights in $\mathbf{w}$, and $\mathbf{x}$ is arranged accordingly. Call $\mathcal{S}_0$, $\mathcal{S}_-$ and $\mathcal{S}_+$ the corresponding sets of indices in $\mathbf{w}$. Now, if $\mathbf{x}$ goes to the right, we represent the feature selected as a binary vector $\mu^+ \in \{0,1\}^d$, containing ones only at $\mathcal{S}_+$. Similarly, if $\mathbf{x}$ goes to the left binary vector $\mu^- \in \{0,1\}^d$, containing ones only at $\mathcal{S}_-$. We call $\mu^+$ and $\mu^-$ the NODE-FEATURES, where location of one represents features selected by $\mathbf{w}$.

We use NODE-FEATURES, to interpret the dataset as follows:

1. For each decision node NODE-FEATURES represents the features related to left and right subtree. By using NODE-FEATURES, we can understand what set of features separates a group of classes.

2. Features associated with a class $k$: for each node in the path from the root to leaf for class $k$ collect NODE-FEATURES, and at the end take logical OR of all NODE-FEATURES. If there is more than one leaf for class $k$, take the union of all the features selected.

3. For features specific to a given input $\mathbf{x}$: repeat the process as above, but only for the leaf containing the input $\mathbf{x}$. Next, keep only those features that are


Figure 5.1: Single instance of each class in Fashion-MNIST dataset.

active in the $\mathbf{x}$.

Since selected features contain the raw pixel value, we can plot them to visualize what concept is captured by these features.

## 5.2 Experiments

### 5.2.1 Datasets description

In order to validate our approach, we test it on three types of image datasets. The first category is the image dataset where input features are raw pixels: Fashion-MNIST [95] and MNIST, where each input is in $\mathbb{R}^{784}$ space. The second category is the deep neural network features. For LeNet5 [51] features, we extract features from the last convolution layer of a pre-trained LeNet5 trained on MNIST. Each input in this dataset is $\mathbb{R}^{800}$ space. Next, for VGG features, we use features from a pre-trained VGG16 [77] network trained on a subset of 16 classes from the Imagenet dataset [21].

| Raw pixels (Fashion-MNIST, 784 features) | | | | | | Total |
|---|---|---|---|---|---|---|
| T-shirt(0) | Trouser(1) | Pullover(2) | Dress(3) | Coat(4) | Sandal(5) | |
| 154 | 166 | 283 | 153 | 293 | 155 | 440 |
| Shirt(6) | Sneaker(7) | Bag(8) | Ankle boot(9) | | | |
| 157 | 299 | 177 | 163 | | | |
| Raw pixels (MNIST, 784 features) | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | |
| 254 | 164 | 329 | 239 | 290 | 212 | 377 |
| 6 | 7 | 8 | 9 | | | |
| 224 | 260 | 293 | 201 | | | |
| Deep net features (LeNet5 features, 800 features) | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | |
| 91 | 173 | 96 | 171 | 138 | 96 | 395 |
| 6 | 7 | 8 | 9 | | | |
| 114 | 191 | 171 | 138 | | | |
| Deep net features (VGG16 features, 8192 features) | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | |
| 352 | 444 | 572 | 710 | 554 | 544 | |
| 6 | 7 | 8 | 9 | 10 | 11 | |
| 836 | 435 | 440 | 426 | 422 | 419 | 2730 |
| 12 | 13 | 14 | 15 | | | |
| 439 | 406 | 451 | 746 | | | |
| Hand-crafted features (Segment, 19 features) | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | |
| 9 | 7 | 11 | 11 | 9 | 8 | 14 |
| 6 | 7 | | | | | |
| 8 | 9 | | | | | |

Table 5.1: Number of feature selected by the sparse oblique tree for different dataset.

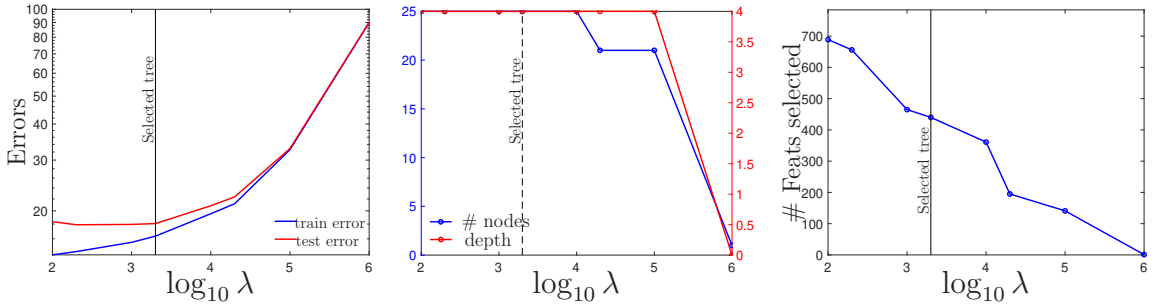


Figure 5.2: Classification error (training and test) and number of nodes and of features selected by the trees as a function of $\lambda$ for Fashion-MNIST dataset. The vertical line indicates the tree we selected ($\lambda = 2000$).

Again we use features from the last convolution layer of the network, and each input is in $\mathbb{R}^{8192}$ space. The final category is the hand-crafted features; here, we use the Segment dataset. It is a UCI [8] dataset where the instances were drawn randomly from a database of 7 outdoor images. The images were hand-segmented to create a classification for every pixel, and each input is in $\mathbb{R}^{19}$ space.

Here, we discuss the interpretibility results on Fashion-MNIST [95] (figure 5.1), and summarize other datasets in the table 5.2 and 5.1.

### 5.2.2 Experiment setup

For each dataset, we choose an initial tree structure for TAO of a certain depth and random initial values for the weights at the nodes. The decision nodes are hyperplanes, and each leaf contains a single class label. We constructed a collection of trees over a range of sparsity parameter $\lambda \in [0, \infty)$. From these trees, we pick the tree that is accurate as well as sparse enough for interpretability. In table 5.2 we report the results of these trees. In the same table, we also report the best accuracy, in which case the tree is deeper and less sparse. Although, we would not use those trees for interpretability.

We also trained a CART tree for each dataset, and as mentioned earlier, as the

| Model | Training Error | Test Error | Height | # Nodes |
|---|---|---|---|---|
| | | Fashion MNIST | | |
| TAO (best) | 5.88 | 14.11 | 8 | 211 |
| TAO | 16.84 | 18.08 | 5 | 25 |
| CART | 2.14 | 20.88 | 30 | 9443 |
| | | MNIST | | |
| TAO (best) | 1.57 | 5.26 | 8 | 177 |
| TAO | 9.23 | 9.58 | 6 | 39 |
| CART | 0.42 | 12.26 | 20 | 6857 |
| | | LeNet5 features | | |
| TAO (best) | 0.02 | 1.78 | 6 | 166 |
| TAO | 2.35 | 2.65 | 6 | 23 |
| CART | 0.31 | 6.56 | 20 | 3165 |
| | | VGG features | | |
| TAO (best) | 0 | 7.62 | 6 | 51 |
| TAO | 2.35 | 2.65 | 6 | 39 |
| CART | 0.31 | 6.56 | 20 | 3165 |
| | | Segment | | |
| TAO | 3.23 | 3.57 | 6 | 27 |
| CART | 0.05 | 5.62 | 14 | 129 |

Table 5.2: Training and test errors, and parameter for CART and TAO on different datasets. If TAO (best) is present in a row, it represents best tree for classification, but we do not use that tree for interpretability.

dataset becomes complicated, the CART tree becomes really difficult to interpret due to a large number of nodes, as shown in table 5.2 and figure 5.5.

### 5.2.3 Interpretability results on Fashion-MNIST

For the Fashion-MNIST dataset, we use the initial tree of depth 5 (total 31 nodes), and train it over a range of sparsity parameter $\lambda$ as mentioned above. In figure 5.2 we show the training/test errors as a function of sparsity parameters. Using these

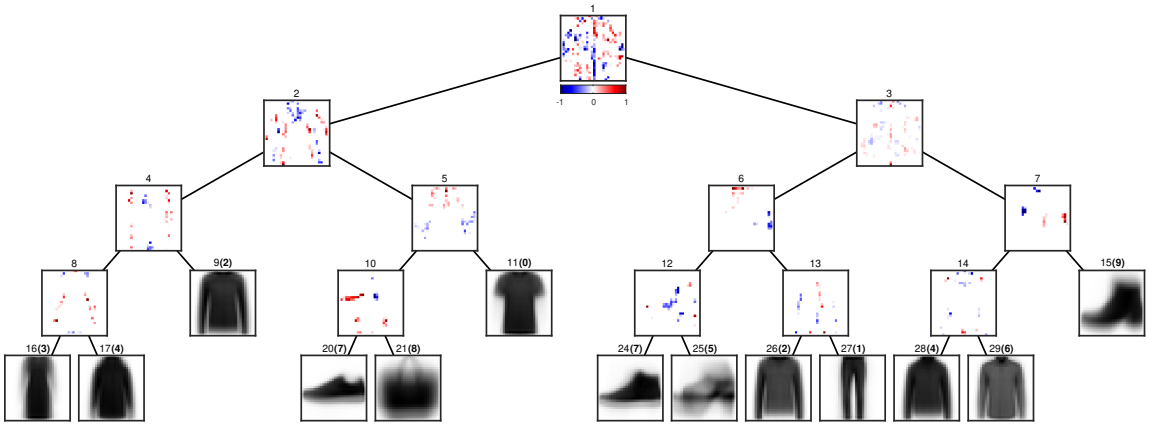


Figure 5.3: Tree selected to interpret the Fashion-MNIST dataset ($\lambda = 2000$). At each decision node, we show its weight vector and node index. At each leaf, we show their index, class label, and the average of training instances reaching the leaf. We plot both the weight vector and the average, of dimension 784, as a $28 \times 28$ square. Weight vectors are colored according to their sign and magnitude (positive, negative and zero values are blue, red, and white, respectively), and averages as greyscale.

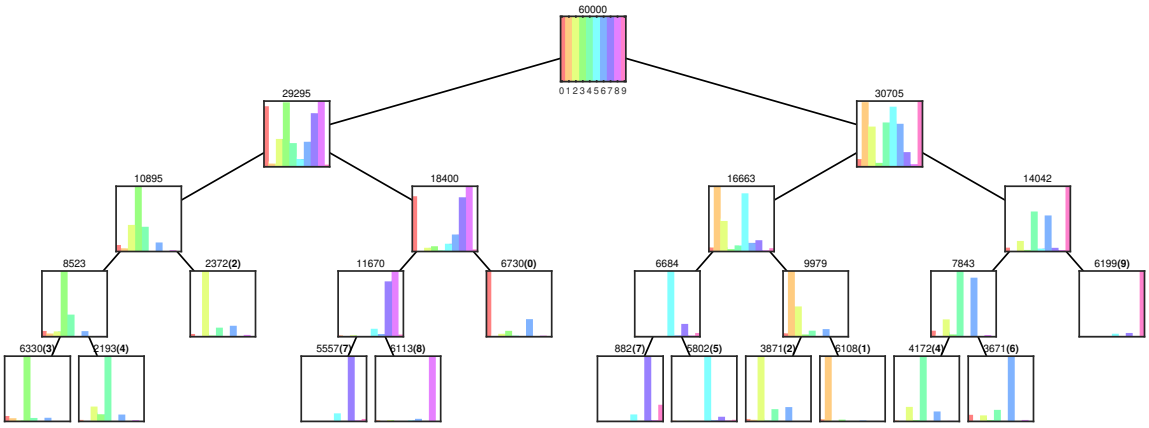


Figure 5.4: Class histograms; we show the number of training instances reaching the node and, for leaves, their label. You may need to zoom in the plot.

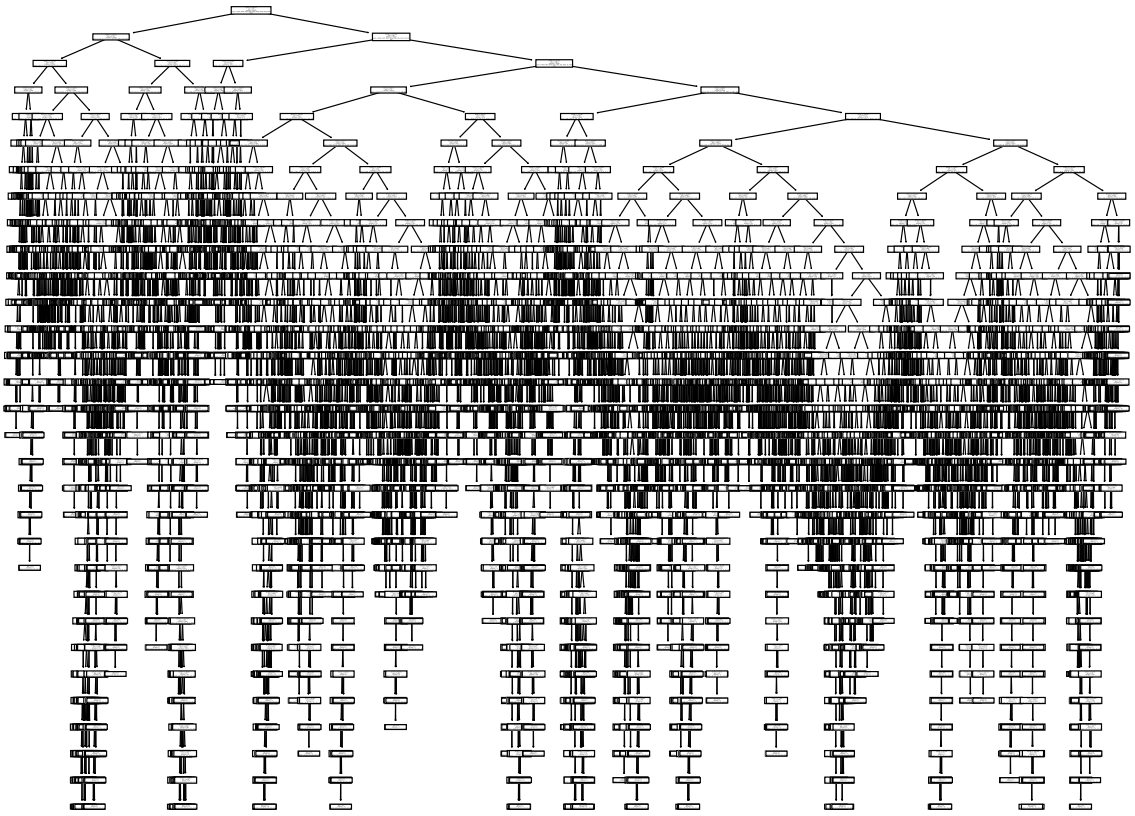

Figure 5.5: CART tree trained on Fashion-MNIST dataset.

plots, we pick the tree with the lowest test error ($\lambda = 2000$). In figure 5.3, we show the weight vector of each decision node, and in figure 5.4 we show the histogram of instances at each node.

### 5.2.4 Sub-groups in sneakers and pullover class

Determining sub-groups within a given class is very important to interpret the dataset. As shown in the figure 5.3, there exist sub-groups within a single class. For instance, both in class sneakers and pullover, there exist two sub-groups in each class. In class sneakers by looking at the average image in the leaves (number 20 and 24),

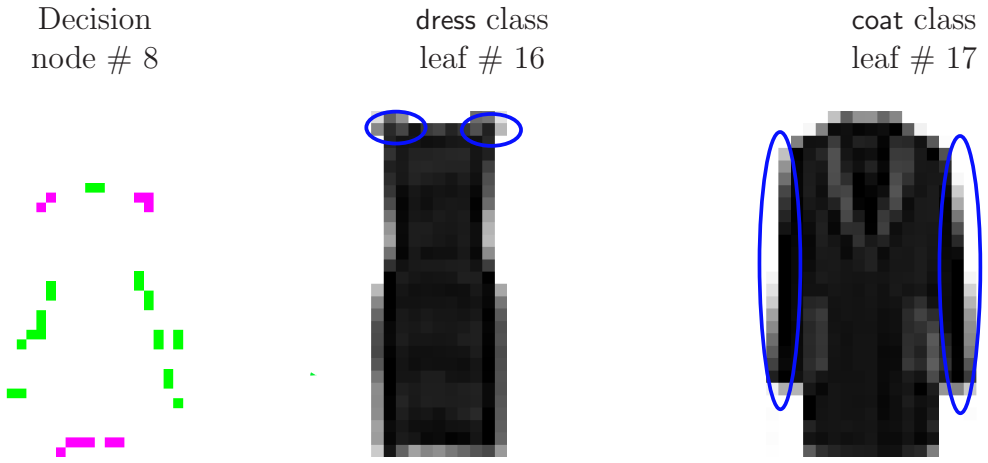


Figure 5.6: *Left:* weight vector of the decision node # 8 plotted as 28×28 image. Magenta color represents negative weights (left child) and green color represents positive weights (right child). *Middle:* one instance of class dress, and blue ellipses show the location of magenta color weights in the image. *Right:* one instance of class coat, and blue ellipses show the location of green color weights.

it is easy to see two types of sneakers in this class. These subgroups are different in terms of the height of the ankle part of the shoe. Since there are so many instances, it is difficult to predict this kind of sub-groupings. However, the sparse oblique trees allow us to visualize these groups easily.

### 5.2.5 Difference between pairs of classes or groups of classes

There is a lot of irrelevant information in a given image in the image dataset, making it hard to understand what concept differentiates the two classes. So, it is very difficult to summarize the difference between classes by just looking at the images. However, by examining the weights of the two classes' decision nodes, we can understand what concept separates the two classes. For instance, as shown in the figure 5.6 the weights of decision node number 8 clearly describe the difference between class dress and coat. The tree mostly focuses on the sleeves; if it is present,


Figure 5.7: Same as figure 5.6, but for a group of classes.

then it is a coat (green weights); otherwise, it is a dress. This makes sense when we look at the instance of the dress class, where most instances do not have long sleeves, unlike to coat class, where all instances have long sleeves.

Similarly, by examining the decision nodes closer to the root, we can determine the difference between a group of classes. We show a few such examples in figure 5.7. As shown in the figure 5.3 node # 5 separates class sneaker and bag to the left side, tshirt to the right side. The decision node separates tshirt from other two, based on the presence of ink on the *neck*, as both bag and sneaker do not have ink on that region. In the second row the decision node # 7 separates the left group (coat and shirt) from the right group (ankle boots) in terms of ink present at the top of the ankle boots. If the ink is present at that location, the node sends the instance to the right child; otherwise, the left child. In third row of the figure 5.7 we see one interesting example, where the tree needs very few pixels to differentiate between two groups: Sneaker and sandals on the left, and pullover and trouser on the right. The node checks if the instance has *collar* or the *belt*, then the instance goes to the right side; otherwise, if the instance has ink at the *right side of shoe*, then it goes to the left. These insights about the data are difficult to attain with any other classifier.

### 5.2.6 Feature selection for a given instance

The proposed approach can also select features that are specific to a given instance. This way, we can track what part of the images is selected at each decision node. This provides an if-else-based rule for a given prediction, creating a set of rules to understand what features are required for a given instance to be classified as a specific class.

We give one such example for an instance of class t-shirt in figure 5.8. As shown in the first node, the tree focuses on the *neck*, left *short sleeves* and the middle pixels in the image. Then for the second node in the path (node # 2), the tree checks for

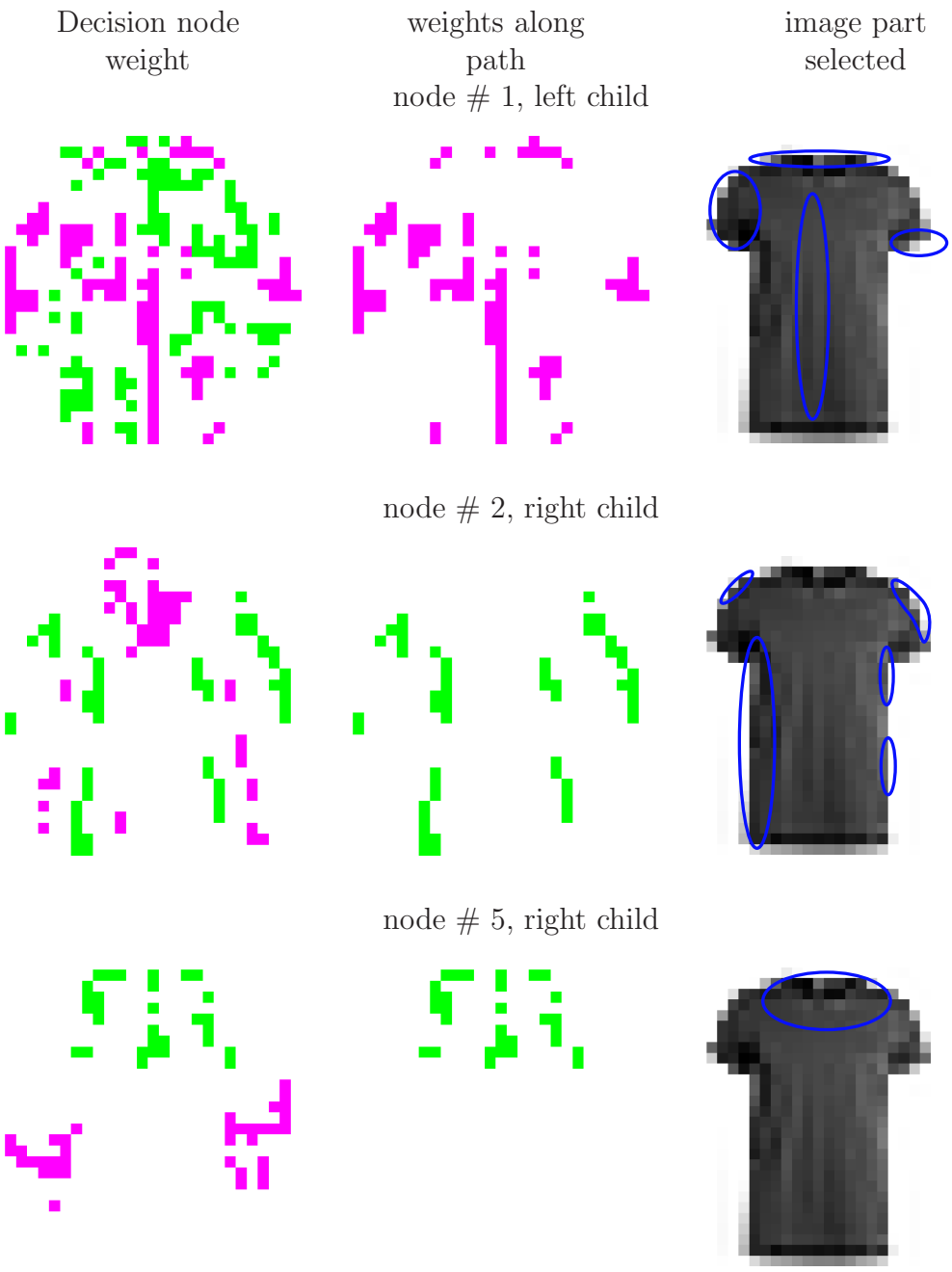


Figure 5.8: *Left:* weight vectors of decision nodes plotted as 28×28 image. Magenta color represents negative weights (left child), and green color represents positive weights (right child). *Middle:* weights along the path from the root to leaf # 11. *Right:* one instance of class t-shirt, and blue ellipses show the location of weights from the middle column in the image.

the shape of the t-shirt to separate it from the other classes (*dress, coat, pullover*). Finally, at node # 5, the tree uses *neck* as the discriminatory features to send the instance at leaf # 11.

## 5.3 Conclusion

In this section, we show how sparse oblique trees can be used to interpret image datasets. Since these trees can achieve high accuracy while remaining interpretable, we can use them to understand what part of an image is relevant for a particular class or sub-groups of a class. By extending our masking operation from the previous chapter, we can establish the relationship between classes and the input features using the weights of the decision nodes. This allows us to understand the underlying difference between classes, sub-groups of a class, or groups of classes. Also, using the hierarchical structure of the oblique trees, we can extract features that are tailored not only to a class but also for specific instances.

# Chapter 6

# Conclusion and Future Work

This dissertation presents two novel approaches to interpret deep neural networks. The first approach focuses on understanding what information is retained by the parameters of the network. We achieve this by generating multiple inverse sets corresponding with different activation ranges for a given neuron in the network. Inspection of these regions by a human can reveal regularities that help us characterize individual neurons' behavior more comprehensively. The second approach discusses what group of neurons are related to a specific class. For this, we mimic the classifier part of the network using sparse oblique decision trees and use the parameters of decision nodes to establish relationships between neurons and classes. We also present a surprising discovery that out of thousands of neurons in the network, only a small subset of neurons are responsible for a given class. Finally, we extend the second approach to interpret image datasets by training sparse oblique trees directly over the pixels of the images and demonstrating what concept (group of features) differentiates between a group of classes or sub-groups within a class.

Our ability to characterize a given neuron's behavior and determine how different neurons in the deep neural network interact with each other to make a decision can lead to several future research directions, either by extending these approaches

individually or as a combination of the two. In the rest of this section, we discuss some of these future research projects.

**Interpreting NLP datasets** In chapter 5, we show how we can use sparse oblique decision trees to interpret image datasets. So, it is natural to explore this approach for other datasets in different domains, such as NLP datasets. We can represent the raw text data into TF-IDF (term frequency-inverse document frequency) format and then apply the same approach as in chapter 5 to find the interesting group of input features. However, unlike image datasets, the features selected by the decision tree are not directly interpretable, as the features are in TF-IDF format. We need to map back the selected features to the corresponding word in the vocabulary, using their original index in a feature vector.

**Extracting class-specific concepts** Another possible direction of future research is to extract class-specific human-interpretable concepts from the deep net using sparse oblique decision trees. We demonstrate in chapter 4 that by mimicking the classifier part of the network using sparse oblique decision trees, we can find the class-specific neurons effectively. We can project these neurons into input space to provide a human-interpretable, rule-based global explanation for the network. One way to achieve this is by projecting selected neurons from each class to the input space. We can use the "sampling the inverse set" approach (chapter 3) here. Instead of generating the inverse set for a single neuron, we can do it for a group of neurons, same as section 3.3.1. We can further fine-tune results by setting different activation ranges for different neurons. Next, we can generate multiple inverse sets with varying activation ranges, allowing us to visualize what concepts in the input change the class probability.

# Appendix A

# Neural Network Architectures

Tables A.1 and A.2 describe the architectures for our LeNet5 network and the VGG16 network used in the experiments in chapters 4–5.

| Block name | Block description |
|---|---|
| conv1 | convolution with kernel size $5 \times 5$ and 20 channels<br>ReLU<br>maxpooling with kernel size $2 \times 2$ |
| conv2 | convolution with kernel size $5 \times 5$ and 50 channels<br>ReLU<br>maxpooling with kernel size $2 \times 2$ |
| fc1 | Fully connected layer with 500 hidden units<br>ReLU |
| dropout | p=0.5 |
| fc2 | Fully connected layer with 10 hidden units |

Table A.1: LeNet5 architecture for MNIST dataset.

| Layer | Connectivity | Layer | Connectivity |
|---|---|---|---|
| Input | 64×64×3 Image | | |
| | | 11 | convolutional, 512 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU |
| 1 | convolutional, 64 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU | | |
| | | 12 | convolutional, 512 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU |
| 2 | convolutional, 64 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU | | |
| | | 13 | convolutional, 512 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU |
| 3 | max pool, 2× 2 window (stride=2) | | |
| 4 | convolutional, 128 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU | 14 | max pool, 2× 2 window (stride=2) |
| | | 15 | convolutional, 512 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU |
| 5 | convolutional, 128 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU | | |
| | | 16 | convolutional, 512 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU |
| 6 | max pool, 2× 2 window (stride=2) | | |
| 7 | convolutional, 256 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU | 17 | convolutional, 512 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU |
| 8 | convolutional, 256 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU | 18 | Dense Layer 4096 neuron followed by ReLU → Dropout (p=0.6) |
| 9 | convolutional, 256 3×3 filters (stride=1,padding = 1) followed by BatchNormalization → ReLU | 19 | Dense Layer 4096 neuron followed by ReLU → Dropout (p=0.6) |
| | | 20 | Dense Layer 16 neuron |
| 10 | max pool, 2× 2 window (stride=2) | 21 | softmax |

Table A.2: Architecture of our modified VGG16 neural net for our ImageNet subset (64× 64 image size).

# Bibliography


[1] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim. Sanity checks for saliency maps. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 9505–9515. MIT Press, Cambridge, MA, 2018.

[2] C. C. Aggarwal. *Data Mining. The Textbook*. Springer-Verlag, 2015.

[3] R. Andrews, J. Diederich, and A. B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, Dec. 1995.

[4] B. Antonio, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Using negotiable features for prescription problems. *Computing*, 91:135–168, Feb. 2011.

[5] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7):e0130140, 2015.

[6] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: quantifying interpretability of deep visual representations. In *Proc. of*

*the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 6541–6549, Honolulu, HI, July 21–26 2017.

[7] A. Bella, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Using negotiable features for prescription problems. *Computing*, 91:135–168, Feb. 2011.

[8] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. 1998.

[9] L. J. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Calif., 1984.

[10] M. Á. Carreira-Perpiñán. The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models. arXiv, 2021.

[11] M. Á. Carreira-Perpiñán and S. S. Hada. Counterfactual explanations for oblique decision trees: Exact, efficient algorithms. In *Proc. of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*, pages 6903–6911, Online, Feb. 2–9 2021.

[12] M. Á. Carreira-Perpiñán and S. S. Hada. Counterfactual explanations for oblique decision trees: Exact, efficient algorithms. arXiv:2103.01096, Mar. 1 2021.

[13] M. Á. Carreira-Perpiñán and S. S. Hada. Inverse classification with logistic and softmax classifiers: Efficient optimization. arXiv, 2021.

[14] M. Á. Carreira-Perpiñán and P. Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 1211–1221. MIT Press, Cambridge, MA, 2018.

[15] M. Á. Carreira-Perpiñán and A. Zharmagambetov. Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting. In *Proc. of the 2020 ACM-IMS Foundations of Data Science Conference (FODS 2020)*, pages 35–46, Seattle, WA, Oct. 19–20 2020.

[16] L. Carrillo-Reid, S. Han, W. Yang, A. Akrouh, and R. Yuste. Controlling visually guided behavior by holographic recalling of cortical ensembles. *Cell*, 178(2):447–457.e5, July 11 2019.

[17] M. Craven and J. W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Proc. of the 11th Int. Conf. Machine Learning (ICML'94)*, pages 37–45, 1994.

[18] M. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 8, pages 24–30. MIT Press, Cambridge, MA, 1996.

[19] Z. Cui, W. Chen, Y. He, and Y. Chen. Optimal action extraction for random forests and boosted trees. In *Proc. of the 21st ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2015)*, pages 179–188, Sydney, Australia, Aug. 10–13 2015.

[20] A. Datta, S. Sen, and Y. Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *IEEE Symposium on Security and Privacy (SP 2016)*, pages 598–617, 2016.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. of the 2009 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'09)*, pages 248–255, Miami, FL, June 20–26 2009.

[22] D. J. Dittman, T. M. Khoshgoftaar, R. Wald, and J. V. Hulse. Comparative analysis of dna microarray data through the use of feature selection techniques. In *9th Int. Conf. Machine Learning and Applications (ICMLA)*, pages 147–152, Washington DC, Dec. 12–14 2010.

[23] P. Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2(1–4):187–202, 1998.

[24] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, Las Vegas, NV, June 26 – July 1 2016.

[25] A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 658–666. MIT Press, Cambridge, MA, 2016.

[26] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, Université de Montréal, June 2009.

[27] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proc. 16th Int. Conf. Computer Vision (ICCV'17)*, pages 3449–3457, Venice, Italy, Dec. 11–18 2017.

[28] G. Forman. An extensive empirical study of feature selection metrics for text classification. *J. Machine Learning Research*, 3:1289–1305, Mar. 2003.

[29] L. Fu. Rule generation from neural networks. *IEEE Trans. Systems, Man, and Cybernetics*, 24(8):1114–1124, Aug. 1994.

[30] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani,

M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 27, pages 2672–2680. MIT Press, Cambridge, MA, 2014.

[31] B. Goodman and S. Flaxman. European Union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, Fall 2017.

[32] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):93, May 2018.

[33] R. Guidotti, A. Monreale, F. Giannotti, D. Pedreschi, S. Ruggieri, and F. Turini. Factual and counterfactual explanations for black box decision making. *IEEE Access*, 34(6):14–23, Nov. – Dec. 2019.

[34] S. S. Hada and M. Á. Carreira-Perpiñán. Sampling the "inverse set" of a neuron: An approach to understanding neural nets. arXiv:1910.04857, Sept. 27 2019.

[35] S. S. Hada and M. Á. Carreira-Perpiñán. Sampling the "inverse set" of a neuron. In *IEEE Int. Conf. Image Processing (ICIP 2021)*, pages 3712–3716, Online, Sept. 19–22 2021.

[36] S. S. Hada and M. Á. Carreira-Perpiñán. Exploring counterfactual explanations for classification and regression trees. In *ECML PKDD 3rd Int. Workshop and Tutorial on eXplainable Knowledge Discovery in Data Mining (XKDD 2021)*, pages 489–504, 2021.

[37] S. S. Hada, M. Á. Carreira-Perpiñán, and A. Zharmagambetov. Sparse oblique decision trees: A tool to understand and manipulate neural net features. arXiv:2104.02922, Apr. 7 2021.

[38] S. S. Hada, M. Á. Carreira-Perpiñán, and A. Zharmagambetov. Understanding and manipulating neural net features using sparse oblique classification trees. In *IEEE Int. Conf. Image Processing (ICIP 2021)*, pages 3707–3711, Online, Sept. 19–22 2021.

[39] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, third edition, 2011.

[40] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, 2001.

[41] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 2015.

[42] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov. 1997.

[43] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093 [cs.CV], June 20 2014.

[44] K. Kanamori, T. Takagi, K. Kobayashi, and H. Arimura. DACE: Distribution-aware counterfactual explanation by mixed-integer linear optimization. In *Proc. of the 20th Int. Joint Conf. Artificial Intelligence (IJCAI'07)*, pages 2855–2862, Hyderabad, India, Jan. 6–12 2007.

[45] M. Karim and R. M. Rahman. Decision tree and naïve bayes algorithm for classification and generation of actionable knowledge for direct marketing. *Journal of Software Engineering and Applications*, 6:196–206, Jan. 2013.

[46] A.-H. Karimi, G. Barthe, B. Balle, and I. Valera. Model-agnostic counterfactual explanations for consequential decisions. In *Proc. of the 23rd Int. Conf. Artificial Intelligence and Statistics (AISTATS 2020)*, pages 895–905, Online, Aug. 26–28 2020.

[47] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 1885–1894, Sydney, Australia, Aug. 6–11 2017.

[48] P. W. Koh, K.-S. Ang, H. H. K. Teo, and P. Liang. On the accuracy of influence functions for measuring group effects. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 32, pages 5254–5264. MIT Press, Cambridge, MA, 2019.

[49] I. E. Kumar, S. Venkatasubramanian, C. Scheidegger, and S. Friedler. Problems with Shapley-value-based explanations as feature importance measures. In H. Daumé III and A. Singh, editors, *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pages 5491–5500, Online, July 13–18 2020.

[50] M. L. Leavitt and A. Morcos. Towards falsifiable interpretability research. arXiv:2010.12016, Oct. 22 2020.

[51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov. 1998.

[52] A. Lucic, H. Oosterhuis, H. Haned, and M. de Rijke. FOCUS: Flexible optimizable counterfactual explanations for tree ensembles. In *Proc. of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, Online, Feb. 22 – Mar. 1 2022.

[53] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In I. Guyon, U. v. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 4765–4774. MIT Press, Cambridge, MA, 2017.

[54] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *Int. J. Computer Vision*, 120(3):233–255, Dec. 2016.

[55] J. H. Marshel, Y. S. Kim, T. A. Machado, S. Quirin, B. Benson, J. Kadmon, C. Raja, A. Chibukhchyan, C. Ramakrishnan, M. Inoue, J. C. Shane, D. J. McKnight, S. Yoshizawa, H. E. Kato, S. Ganguli, and K. Deisseroth. Cortical layer-specific critical dynamics triggering perception. *Science*, 365(6453): eaaw5202, Aug. 9 2019.

[56] D. Martens and F. Provost. Explaining data-driven document classifications. *MIS Quarterly*, 38:73–100, Mar. 2014.

[57] K. McCormick, D. Abbott, M. S. Brown, T. Khabaza, and S. R. Mutchler. *IBM SPSS Modeler Cookbook*. Packt Publishing, 2013.

[58] L. Merrick and A. Taly. The explanation game: Explaining machine learning models using Shapley values. In *Int. Cross-Domain Conf. for Machine Learning and Knowledge Extraction (CD-MAKE 2020)*, pages 17–38, 2020.

[59] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks, June 17 2015.

[60] J. Mu and J. Andreas. Compositional explanations of neurons. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 33, pages 17153–17163. MIT Press, Cambridge, MA, 2020.

[61] W. J. Murdoch, P. J. Liu, and B. Yu. Beyond word importance: Contextual decomposition to extract interactions from LSTMs. In *Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018)*, Vancouver, Canada, Apr. 30 – May 3 2018.

[62] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, pages 427–436, Boston, MA, June 7–12 2015.

[63] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 3387–3395. MIT Press, Cambridge, MA, 2016.

[64] A. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[65] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 3510–3520, Honolulu, HI, July 21–26 2017.

[66] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

[67] A. Parmentier and T. Vidal. Optimal counterfactual explanations in tree en-

sembles. In M. Meila and T. Zhang, editors, *Proc. of the 38th Int. Conf. Machine Learning (ICML 2021)*, pages 8422–8431, Online, July 18–24 2021.

[68] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, Aug. 2005.

[69] G. Pruthi, F. Liu, M. Sundararajan, and S. Kale. Estimating training data influence by tracing gradient descent. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 33, pages 19920–19930. MIT Press, Cambridge, MA, 2020.

[70] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[71] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proc. of the 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2016)*, pages 1135–1144, San Francisco, CA, Aug. 13–17 2016.

[72] M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, pages 1527–1535, New Orleans, LA, Feb. 2–7 2018.

[73] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Machine Intelligence*, 1 (5):206–215, May 2019.

[74] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual explanations from deep networks via gradient-based local-

ization. In *Proc. 16th Int. Conf. Computer Vision (ICCV'17)*, pages 618–626, Venice, Italy, Dec. 11–18 2017.

[75] L. S. Shapley. *A Value for n-Person Games*. Princeton University Press, 1953.

[76] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 3145–3153, Sydney, Australia, Aug. 6–11 2017.

[77] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.

[78] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. of the 2nd Int. Conf. Learning Representations (ICLR 2014)*, Banff, Canada, Apr. 14–16 2014.

[79] C. Singh, W. J. Murdoch, and B. Yu. Hierarchical interpretations for neural network predictions. In *Proc. of the 7th Int. Conf. Learning Representations (ICLR 2019)*, New Orleans, LA, May 6–9 2019.

[80] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for Simplicity: The All Convolutional Net. In *ICLR (workshop track)*, 2015.

[81] E. Štrumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3):647–665, 2014.

[82] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. arXiv:1703.01365, Mar. 17 2017.

[83] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *Proc. of the 2nd Int. Conf. Learning Representations (ICLR 2014)*, Banff, Canada, Apr. 14–16 2014.

[84] J. J. Thiagarajan, B. Kailkhura, P. Sattigeri, and K. N. Ramamurthy. TreeView: Peeking into Deep Neural Networks Via Feature-Space Partitioning. In *Proceedings of the NIPS Workshop on Interpretable Machine Learning for Complex Systems*, 2016.

[85] G. Tolomei, F. Silvestri, A. Haines, and M. Lalmas. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *Proc. of the 23rd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2017)*, pages 465–474, Halifax, Nova Scotia, Aug. 13–17 2017.

[86] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, Oct. 1993.

[87] B. Ustun, A. Spangher, and Y. Liu. Actionable recourse in linear classification. In *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAT 2019)*, pages 10–19, Atlanta, GA, Jan. 29–31 2019.

[88] A. Van Looveren and J. Klaise. Interpretable counterfactual explanations guided by prototypes. In N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J. A. Lozano, editors, *Proc. of the 32nd European Conf. Machine Learning (ECML–21)*, Bilbao, Spain, Sept. 13–17 2021.

[89] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard J. Law & Technology*, 31(2):841–887, Spring 2018.

[90] D. Wei, B. Zhou, A. Torralba, and W. Freeman. Understanding intra-class knowledge inside CNN. arXiv:1507.02379, July 15 2015.

[91] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viégas, and J. Wilson. The What-If Tool: Interactive probing of machine learning models. *IEEE Trans. Visualization and Computer Graphics*, 26(1):56–65, Jan. 2020.

[92] A. White and A. d'Avila Garcez. Measurable counterfactual local explanations for any classifier. In G. D. Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, editors, *Proc. 24th European Conf. Artificial Intelligence (ECAI 2020)*, pages 2529–2535, Aug. 29 – Sept. 8 2020.

[93] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, fourth edition, 2016.

[94] Z. Xia, C. Zhu, Z. Wang, Q. Guo, and Y. Liu. Every filter extracts a specific texture in convolutional neural networks. arXiv:1608.04170, Aug. 16 2016.

[95] H. Xiao, K. Rasula, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747, 15 2017.

[96] Q. Yang, J. Yin, C. X. Ling, and R. Pan. Extracting actionable knowledge from decision trees. *IEEE Trans. Knowledge and Data Engineering*, 18(1): 43–56, Jan. 2006.

[97] C.-K. Yeh, J. S. Kim, I. E. H. Yen, and P. Ravikumar. Representer point selection for explaining deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31. MIT Press, Cambridge, MA, 2018.

[98] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. In *ICML Workshop on Deep Learning*, 2015.

[99] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. arXiv:1506.06579, June 15 2015.

[100] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study. *PLoS Medicine*, 15 (11):e1002683, Nov. 2018.

[101] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proc. 13th European Conf. Computer Vision (ECCV'14)*, pages 818–833, Zürich, Switzerland, Sept. 6–12 2014.

[102] Q. Zhang, Y. Yang, H. Ma, and Y. N. Wu. Interpreting CNNs via decision trees. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 6261–6270, Long Beach, CA, June 16–20 2019.

[103] A. Zharmagambetov and M. Á. Carreira-Perpiñán. Smaller, more accurate regression forests using tree alternating optimization. In H. Daumé III and A. Singh, editors, *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pages 11398–11408, Online, July 13–18 2020.

[104] A. Zharmagambetov, M. Gabidolla, and M. Á. Carreira-Perpiñán. Improved boosted regression forests through non-greedy tree optimization. In *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021.

[105] A. Zharmagambetov, M. Gabidolla, and M. Á. Carreira-Perpiñán. Improved multiclass AdaBoost for image classification: The role of tree optimization. In *IEEE Int. Conf. Image Processing (ICIP 2021)*, pages 424–428, Online, Sept. 19–22 2021.

[106] A. Zharmagambetov, S. S. Hada, M. Gabidolla, and M. Á. Carreira-Perpiñán. Non-greedy algorithms for decision tree optimization: An experimental comparison. In *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021.

[107] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 14. MIT Press, Cambridge, MA, 2002.

[108] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene CNNs. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.

[109] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, Las Vegas, NV, June 26 – July 1 2016.