Sparse Oblique Decision Trees: A Tool to Understand and Manipulate Neural Net Features*

Suryabhan Singh Hada[1],*, Miguel Á. Carreira-Perpiñán[2] and
Arman Zharmagambetov[2]

[1]LinkedIn, work done while at Dept. of Computer Science & Engineering, University of California, Merced
[2]Dept. of Computer Science & Engineering, University of California, Merced

Abstract

The widespread deployment of deep nets in practical applications has lead to a growing desire to understand how and why such black-box methods perform prediction. Much work has focused on understanding what part of the input pattern (an image, say) is responsible for a particular class being predicted, and how the input may be manipulated to predict a different class. We focus instead on understanding what internal features computed by the neural net are responsible for a particular class. We achieve this by mimicking part of the net with a decision tree having sparse weight vectors at the nodes. We are able to learn trees that are both highly accurate and interpretable, so they can provide insights into the deep net black box. Further, we show we can easily manipulate the neural net features in order to make the net predict, or not predict, a given class, thus showing that it is possible to carry out adversarial attacks at the level of the features. We demonstrate this robustly in MNIST and ImageNet with LeNet5 and VGG networks.

Keywords

Decision trees, Deep neural networks, Interpretability

1. Introduction

Deep neural nets are accurate black-box models. They are highly successful in terms of predictive performance but remarkably difficult to understand in terms of how exactly they come up with a prediction. These issues have been known to researchers and practitioners for many years, but it is in the 2010s that deep learning has achieved a wild, unexpected success that has attracted widespread attention beyond computer science. Thus making it urgent to understand the behavior of these models in explanatory terms.

Much work in this regard seeks to understand what a specific neuron in a deep net does. This includes work on finding input patterns that invert the activation of a neuron [2, 3, 4] or maximally activate it [5, 6, 7]; and work that finds input patterns, or parts of them (such as image regions) that have an important effect on the output class, essentially a sensitivity

HI-AI@KDD, Human-Interpretable AI Workshop at the KDD 2024, 26^{th} of August 2024, Barcelona, Spain

[*] Detailed version of this paper appears as [1].

[*] Corresponding author.

shada@ucmerced.edu (S. S. Hada); mcarreira-perpinan@ucmerced.edu (M. Á. Carreira-Perpiñán); azharmagambetov@ucmerced.edu (A. Zharmagambetov)

analysis via gradients or other measure of saliency [6, 8, 7, 9, 10, 11].

Other work seeks to replace a deep net with a simpler, interpretable model that can then be inspected, such as a decision tree, a set of rules or a (sparse) linear model. This can be done locally around an instance [12] or globally for all instances—which is much harder since an interpretable model like decision trees cannot generally approach the accuracy of the deep net [13, 14, 15, 16, 17, 18, 19, 20]. The fundamental problem with these approaches is that traditional decision tree learning algorithms such as CART [21] or C4.5 [22] are unable to learn small yet accurate enough trees to be useful mimicks of a neural net except in very small problems.

Our paper has two contributions that can improve our ability to explain and manipulate trained deep nets. Firstly, we propose decision trees as a tool to understand deep nets. As mentioned above 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 [23, 24] 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. It has been shown to outperform existing tree algorithms such as CART [21] or C4.5 [22] by a large margin [25], and to improve forest [26, 27].

Second, we apply the tree to an internal layer of the deep net, hence mimicking its remaining (classifier) layers, rather than attempting to mimick the entire deep net. *This allows us to study the relation between deep net features (neuron activations) and output classes (unlike the work cited in the second paragraph, which studies the relation between input features and neuron activations).* 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.

Next, we describe how we use trees to understand and manipulate deep net features (section 2), and demonstrate this in MNIST and ImageNet [28] with LeNet5 and VGG16 [29] deep nets (section 4).

2. Sparse oblique trees as a tool to observe a deep neural net

Our overall approach is as follows (see fig. 3 in the appendix). 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 (see details in [23, 24]) 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 have close to highest validation accuracy and is as sparse as possible.

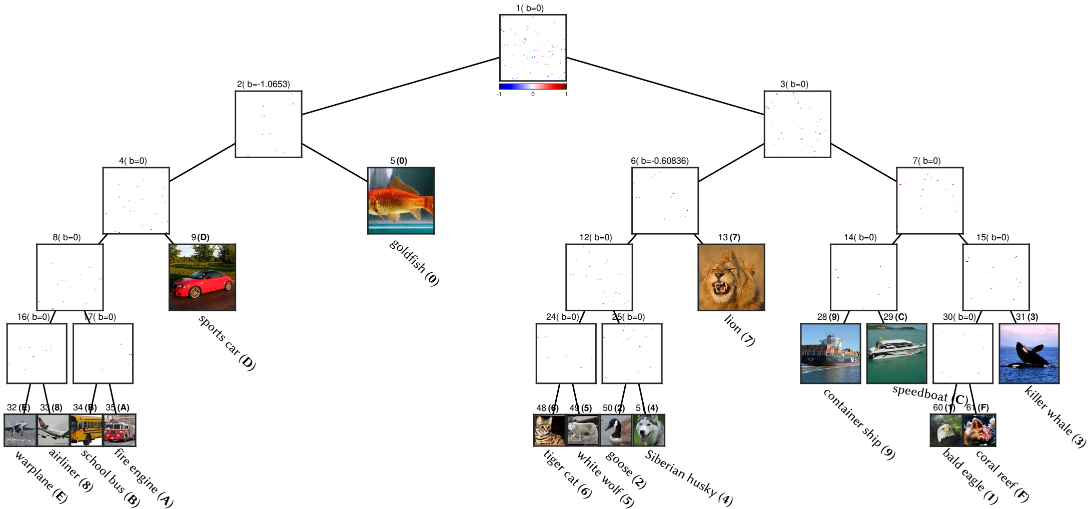

Figure 1: Tree having one leaf per class ($\lambda = 33$). At each decision node we show its weight vector, node index and bias (always zero). At each leaf we show their index, class label, an image of from their class and class description in the format: class description (class label). We plot the 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).

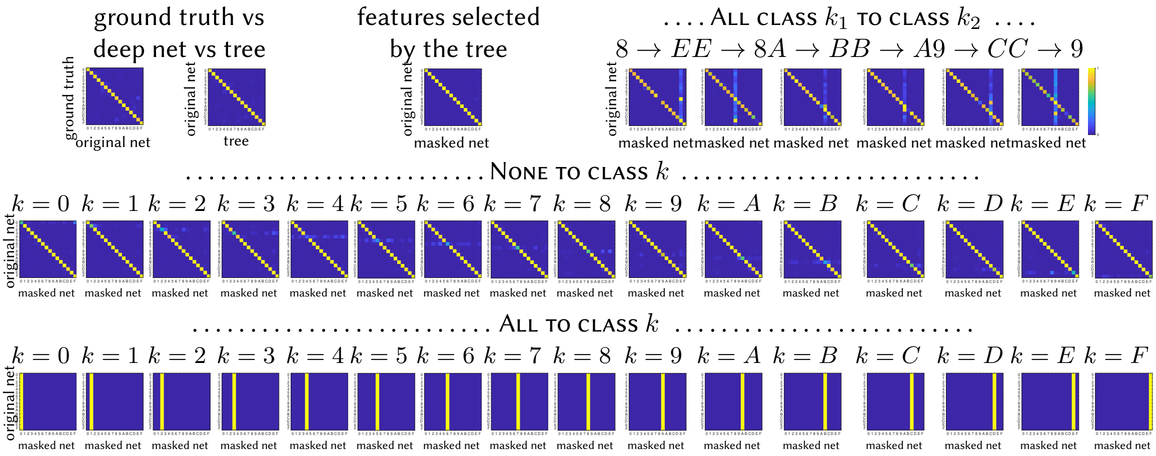

Figure 2: Confusion matrices for VGG (test set). *Top left*: ground-truth vs deep net, and deep net vs tree. *Top middle*: deep net vs deep net with only the features selected by the tree. *Top right*: All class k_1 to class k_2 (selected examples). *Middle*: None to class k. *Bottom*: All to class k.The confusion matrices for the training set (not shown) are very similar.

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

Our goal is to achieve a tree that both mimicks 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 or more globally. Here, we focus on one specific pattern described next.

3. 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).

$$\text{Original net: } \mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{F}(\mathbf{x})) \tag{1}$$

$$\text{Original features: } \mathbf{z} = \mathbf{F}(\mathbf{x}) \tag{2}$$

$$\text{Masked net: } \overline{\mathbf{y}} = \overline{\mathbf{f}}(\mathbf{x}) = \mathbf{g}(\boldsymbol{\mu}(\mathbf{F}(\mathbf{x}))) \tag{3}$$

$$\text{Masked features: } \overline{\mathbf{z}} = \boldsymbol{\mu}(\mathbf{F}(\mathbf{x})) = \boldsymbol{\mu}(\mathbf{z}) \tag{4}$$

In the simplest, most intuitive version of the mask, we just need a binary multiplicative mask $\overline{\mathbf{z}} = \boldsymbol{\mu}^{\times} \odot \mathbf{z}$ where $\boldsymbol{\mu}^{\times} \in \{0,1\}^F$. Using an additive mask and real-valued masks makes the manipulation's effect more robust and harder to detect. We will construct a mask by inspecting the tree, specifically by observing the weight of each feature in each decision node. By selectively zeroing some features we can guarantee that any instance will follow a specific child in a given node and hence direct instances towards a target leaf.

3.1. All instances to one child

We define 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 divert 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 $\mathbf{w}^T\mathbf{z} + b < 0$ 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] The bias (b_i) at each decision node i of the tree is zero. This holds very well in the trees we trained, specifically $|b_i| \ll \|\mathbf{w}_i\|$ at each decision node.

[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.

3.2. Masks

We now show how to construct masks that effect a certain class outcome. For each case, we state the desired goal and the corresponding mask. In the manipulations below we may use NODE-MASK repeatedly over several nodes to construct the mask (which is applied to the feature vector and hence applies globally to each node). In that case, we will only use the multiplicative mask produced by NODE-MASK at each node, and create the additive mask at the end given the final multiplicative mask.

ALL CLASS k_1 TO CLASS k_2: *let* $k_1 \neq k_2 \in \{1, \dots, K\}$. *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 (have the same parent). Class k_2 may be represented by multiple leaves since we only need to deal with one of them (the sibling of k_1). *Mask*: simply apply NODE-MASK to the parent of the leaves of k_1 and k_2. For instance, if class k_1 is left child, then final multiplicative mask $\boldsymbol{\mu}^\times$ will contain ones at $\mathcal{S}_+$, zeros at $\mathcal{S}_-$ and $*$ (meaning any value) at $\mathcal{S}_0$.

NONE TO CLASS k: *let* $k \in \{1, \dots, K\}$. *For any instance originally classified as* k, *classify it as any other class. For any other instance, do not alter its classification. Mask*: simply apply NODE-MASK to the parent of each leaf of k and combine the resulting multiplicative masks as extended-AND (defined below). Finally, add the additive mask.

ALL TO CLASS k: *let* $k \in \{1, \dots, K\}$. *Classify all instances* $\mathbf{x}$ *as class* k. *Mask*: find the path from the root to the leaf of class k. At each node i in the path, apply NODE-MASK (to divert instances along the path) and keep the multiplicative mask only. The final multiplicative mask, elementwise, has a 0 where any of the node masks has a 0, a 1 where all node masks have no 0s but at least one 1, and $*$ elsewhere. This masks out all the "undesired" features that might divert us from the path. Equivalently, this is the logical extended-AND of all the multiplicative masks along the path (where we extend AND to mean $\text{AND}(*, 0) = 0$, $\text{AND}(*, 1) = 1$ and $\text{AND}(*, *) = *$).

4. Experiments

We have evaluated our masks thoroughly on two deep nets. 1) VGG16 [29] in a subset of 16 classes of ImageNet [28], for which we select the $F = 8\,192$ neurons from its last convolutional layer. 2) LeNet5 in MNIST on 10 digit classes [30], for which we select the $F = 800$ neurons at layer conv2 as features. For both of them, we can train trees that accurately mimick the deep net classifier $\mathbf{g}$. The trees give remarkable insight in the relation of deep net features to classes and allow us to construct masks that indeed work as intended in the deep net for most instances. Here, we focus on VGG16.

Our VGG16 net achieves an error of 0.2% (training) and 6.79% (test). To train the tree, we use as initial tree a deep enough, complete binary tree with random parameters, and run TAO for a range of increasing λ values. 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 mimick. 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 normalize the final tree so each node weight vector has norm 1. We also discuss a tree of somewhat lower accuracy but which has exactly one leaf per class (fig. 1). 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).

4.1. Manipulating the deep net features via masks

We derive masks using the mimick tree ($\lambda = 1$). Fig. 2 shows confusion matrices for VGG16, over test instances (see fig. 5 in the appendix for training instances). As shown in the confusion matrix for deep net vs tree prediction (second matrix in the top left), both models have the same prediction for almost all instances, showing the tree mimick the network really well. This was expected as the tree have training and test errors close to those of VGG16. The interesting confusion matrix is the original network vs network with only the feature selected by the tree (top middle). Here, even after using only 17% of the features, the network has the same prediction as to the original one. It suggests that 83% 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. This is not surprising if one notes that deep nets (at least, as presently designed) seem to be vastly overparameterized and can be significantly compressed.

Generally, the masks affect the deep net classification in the same way as the tree. This is to be expected since the tree has a very similar error and confusion matrix as the net, but it is still surprising in how well it works in most cases, like for classifying all instances as class k mask (bottom row). This also indicates that certain deep net neurons (those critically involved in the masks) play a well-defined role in the classification. The number of features that a mask critically needs to perform its job is very small, around 200 (out of 8 192); for MNIST (see fig. 6 and 7 in the appendix) it is much smaller, around 40 (out of 800). Misclassifying class k_1 as k_2 (where k_1 must have a single leaf which is a sibling of k_2) works well too (top right), although a few instances from other classes are sometimes classified as k_2. Not classifying any instance as class k (middle row) works also well but fails with some instances, which remain as class k. The confusion matrices for MNIST (not shown) are very similar. We also demonstrate this masking operation on an image which is not in the dataset for the VGG16 network in the appendix.

4.2. Inspecting the sparse oblique trees

Fig. 1 shows a very interesting tree, obtained for a larger λ value so that there is exactly one leaf per class (the smallest number of leaves possibly unless we ignore classes). This tree has very few nonzero weights yet its test error is reasonable, so it probably extracts features that robustly classify most images. Also, its structure remains unchanged for a wide range of λ. *Inspecting it shows an intuitive hierarchy of classes that seem primarily related to the background or surroundings of the main object in the image.* Its leftmost subtree {warplane, airliner, school bus, fire engine, sports car} consists of man-made objects often found on roads. However, {container ship, speedboat} (man-made objects found on the sea) appears in the rightmost subtree, together with {killer whale, bald eagle, coral reef}, all of which are also typically found on the sea or on the air. Yet {goldfish} appears in a single subtree quite separate from all other classes: indeed, this fish is found on fishbowls (not the sea) in the training images. A subtree in the middle contains animals in land natural environments (forest, snow, grass, etc.): {tiger cat, white wolf, goose, Siberian husky, lion}. And so on. 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 [12, 31]. It points to a possible vulnerability of the net, in that it may misclassify an object that happens to appear in an unusual background (say, a bald eagle standing on a road).

5. Conclusion

Our paper demonstrates the use of sparse oblique decision trees as a powerful "microscope" to investigate the behavior of deep nets, by learning interpretable yet accurate trees that mimick the classifier part of a deep net. Using the TAO algorithm is critical for this to succeed. The resulting tree gives insights about the relation between neurons and classes, and enables the design of simple manipulations of the neuron activations that can, for any training or test instance, change the class predicted in various, controllable ways (thus making adversarial attacks possible at the level of the deep net features).

References

[1] S. S. Hada, M. Á. Carreira-Perpiñán, A. Zharmagambetov, Sparse oblique decision trees: A tool to understand and manipulate neural net features, Data Mining and Knowledge Discovery (2023).

[2] M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: Proc. 13th European Conf. Computer Vision (ECCV'14), Zürich, Switzerland, 2014, pp. 818–833.

[3] A. Dosovitskiy, 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, 2016.

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

[5] D. Erhan, Y. Bengio, A. Courville, P. Vincent, Visualizing Higher-Layer Features of a Deep Network, Technical Report 1341, Université de Montréal, 2009.

[6] K. Simonyan, A. Vedaldi, 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, 2014.

[7] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, 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, R. Garnett (Eds.), Advances in Neural Information Processing Systems (NIPS), volume 29, MIT Press, Cambridge, MA, 2016, pp. 3387–3395.

[8] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, K.-R. Müller, Explaining nonlinear classification decisions with deep Taylor decomposition, Pattern Recognition 65 (2016) 211–222.

[9] R. C. Fong, A. Vedaldi, Net2Vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks, in: Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18), Salt Lake City, UT, 2018, pp. 8730–8738.

[10] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-CAM: Visual explanations from deep networks via gradient-based localization, in: Proc. 16th Int. Conf. Computer Vision (ICCV'17), Venice, Italy, 2017, pp. 618–626.

[11] A. Shrikumar, P. Greenside, A. Kundaje, Learning important features through propagating activation differences, in: D. Precup, Y. W. Teh (Eds.), Proc. of the 34th Int. Conf. Machine Learning (ICML 2017), Sydney, Australia, 2017, pp. 3145–3153.

[12] M. T. Ribeiro, S. Singh, 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), San Francisco, CA, 2016, pp. 1135–1144.

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

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

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

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

[17] G. G. Towell, J. W. Shavlik, Extracting refined rules from knowledge-based neural networks, Machine Learning 13 (1993) 71–101.

[18] M. Craven, 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), 1994, pp. 37–45.

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

[20] P. Domingos, Knowledge discovery via multiple models, Intelligent Data Analysis 2 (1998) 187–202.

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

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

[23] M. Á. Carreira-Perpiñán, 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, R. Garnett (Eds.), Advances in Neural Information Processing Systems (NEURIPS), volume 31, MIT Press, Cambridge, MA, 2018, pp. 1211–1221.

[24] M. Á. Carreira-Perpiñán, The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models, 2022. ArXiv.

[25] A. Zharmagambetov, S. S. Hada, M. Á. Carreira-Perpiñán, M. Gabidolla, An experimental comparison of old and new decision tree algorithms, 2020. ArXiv:1911.03054.

[26] M. Á. Carreira-Perpiñán, 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), Seattle, WA, 2020, pp. 35–46.

[27] A. Zharmagambetov, M. Á. Carreira-Perpiñán, Smaller, more accurate regression forests using tree alternating optimization, in: H. Daumé III, A. Singh (Eds.), Proc. of the 37th Int. Conf. Machine Learning (ICML 2020), Online, 2020, pp. 11398–11408.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, 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), Miami, FL, 2009, pp. 248–255.

[29] K. Simonyan, 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, 2015.

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

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

A. Classifier mimicking

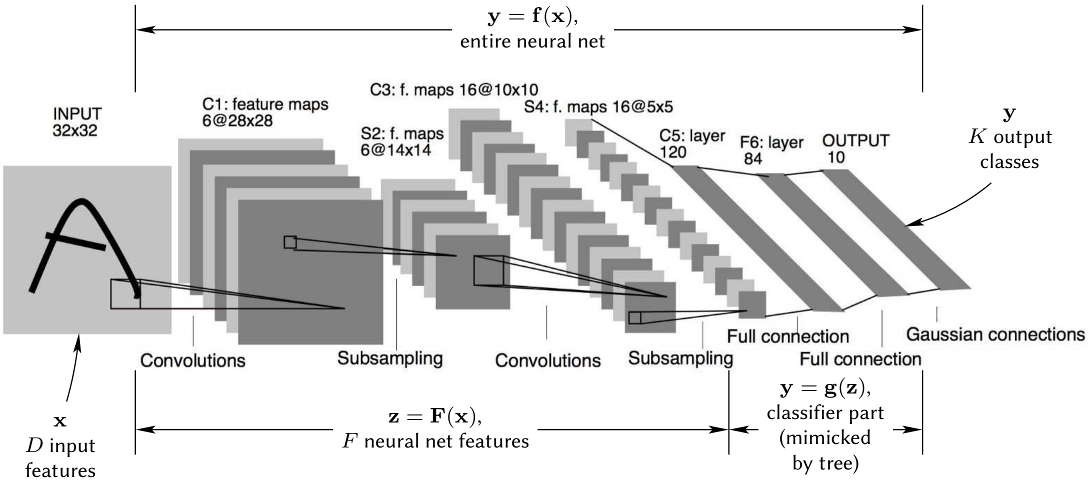

Figure 3: Mimicking part of a neural net with a decision tree. The figure shows the neural net $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{F}(\mathbf{x}))$, considered as the composition of a feature extraction part $\mathbf{z} = \mathbf{F}(\mathbf{x})$ and a classifier part $\mathbf{y} = \mathbf{g}(\mathbf{z})$. For example, for the LeNet5 neural net of [30] 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.

B. Tree to mimic LeNet5 features

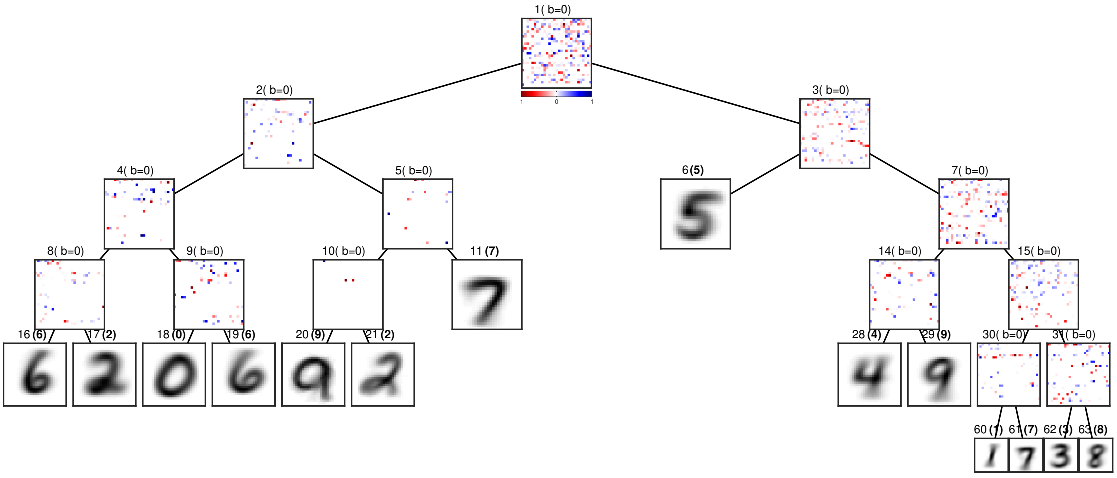

Figure 4: Tree selected as mimic for LeNet5 features ($\lambda = 20$). At each decision node we show weight vector and average of training instances at each leaf; we show the node index, bias (always zero) and, for leaves, their label. We plot the weight vector, of dimension 800, as a 29×29 square (the 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.For MNIST, our LeNet5 architecture achieves an error of 0.00545% (training) and 0.61% (test). We selected as mimick the tree for $\lambda = 20$, with 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.

C. Confusion matrices

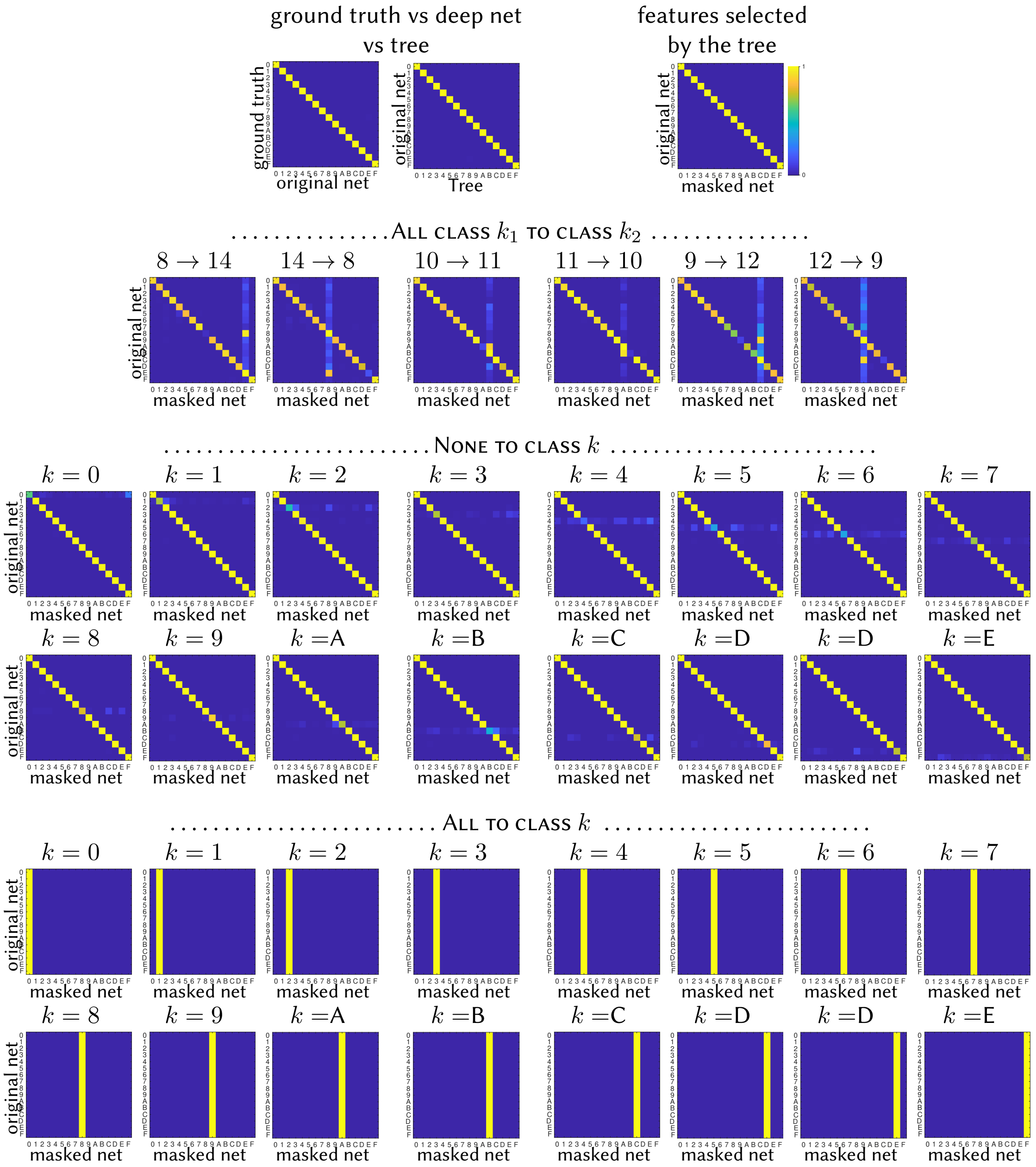

Figure 5: Like fig. 2 but for the training set.

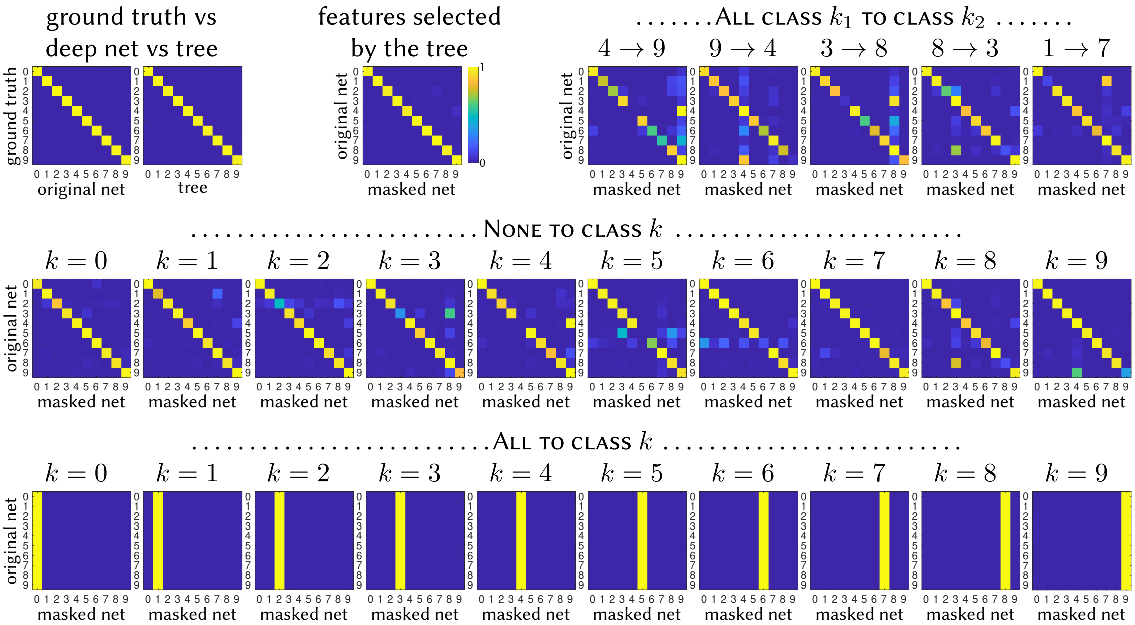

Figure 6: Confusion matrices for LeNet5 (test set). *Top left*: ground-truth vs deep net, and deep net vs tree. *Top middle*: deep net vs deep net with only the features selected by the tree. *Top right*: All class k_1 to class k_2 (selected examples). *Middle*: None to class k. *Bottom*: All to class k.

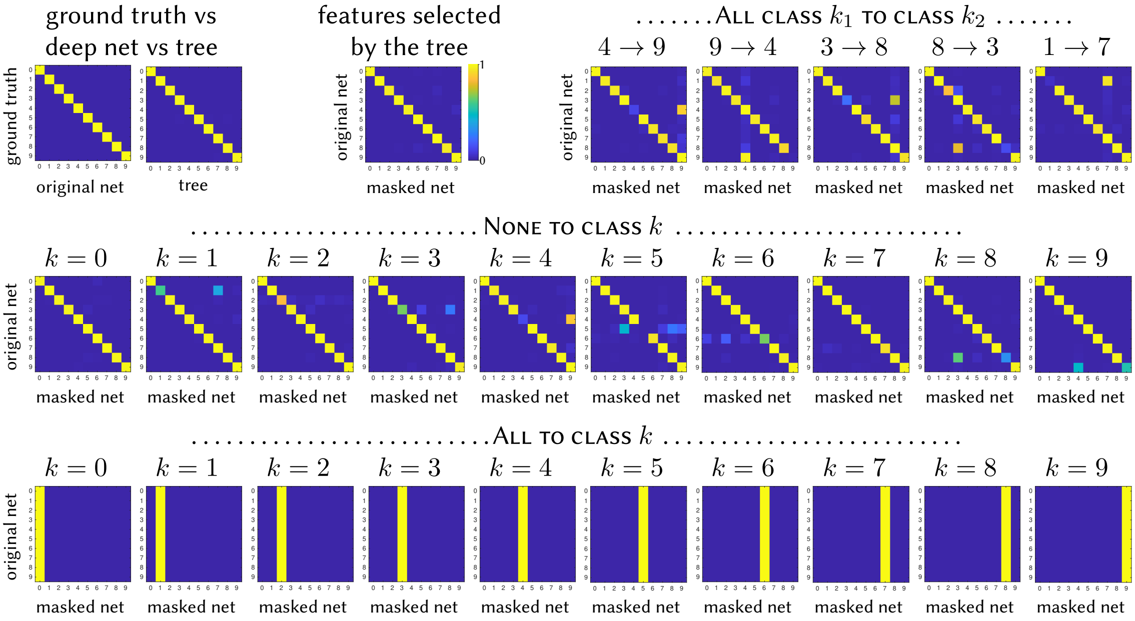

Figure 7: Like fig. 6 but for the training set.

D. Illustration of the masks with an actual image

Fig. 8 illustrates the mask behavior in an image not in the dataset. The middle column histograms show the deep net features (grouped by class). In each row, the top histogram shows the feature values, and the bottom histogram shows the number of features selected for each class. Next, we show how masking the features drastically alters in a controlled way the softmax output. In row 2, when we apply the All to class "Siberian husky" mask, the network now classifies the image as "siberian husky". Similarly, in row 5, when we apply the All to class "bald Eagle" mask, the network now classifies the original image as "bald eagle" with large confidence, compared to row 1, where without the mask the softmax value for "bald eagle" is close to zero. We also show how the mask correlates with superpixels (perceptual groups of pixels obtained by oversegmentation) in the image, either manually cropped (row 3) or optimized to invert the desired deep net features (row 4).

To obtain results like those above, the general procedure is as follows. Firstly (in an offline phase), we train the tree mimic and construct a subset of features $\mathcal{S}_k$ for each class k, using the All to class k mask. This defines a score for an input image $\mathbf{x}$ as $s_k(\mathbf{x}) = \sum_{i \in \mathcal{S}_k} F_i(\mathbf{x})$, where $F_i(\mathbf{x})$ is the feature i computed by the deep neural net for $\mathbf{x}$. We can then discard the tree and the classifier part of the deep net. All we need is the feature-extraction part of the deep net and the class sets $\mathcal{S}_1, \dots, \mathcal{S}_K$.

Then (in an online phase), given an input image and a target class k, we split the image into superpixels (using some oversegmentation algorithm), compute the score for each superpixel, and report the superpixels with lowest score (most salient).

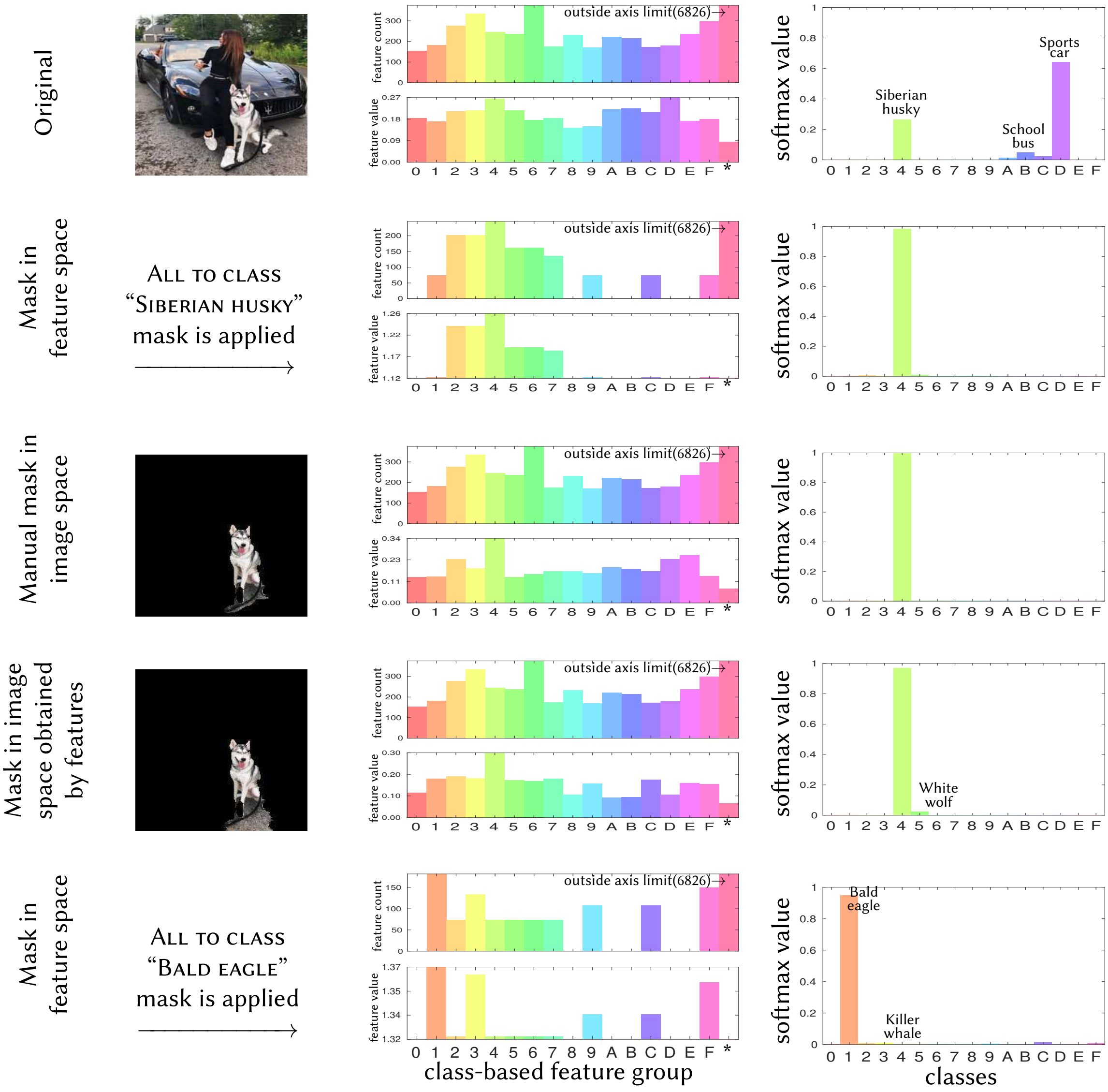

Figure 8: Illustration of masks for a particular image (VGG16 network on ImageNet subset). Column 1 shows the image masks (when available). Column 2 summarizes the 8 192 feature values as two histograms: on the upper panel, the number of features in each class group (listed in the X axis as 0–F, where "∗" 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".