WHAT'S IN THE BOX? EXPLORING THE INNER LIFE OF NEURAL NETWORKS WITH ROBUST RULES

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a novel method for exploring how neurons within a neural network interact. In particular, we consider activation values of a network for given data, and propose to mine noise-robust rules of the form $X \rightarrow Y$, where X and Y are sets of neurons in different layers. To ensure we obtain a small and non-redundant set of high quality rules, we formalize the problem in terms of the Minimum Description Length principle, by which we identify the best set of rules as the one that best compresses the activation data. To discover good rule sets, we propose the unsupervised EXPLAINN algorithm. Extensive evaluation shows that our rules give clear insight in how networks perceive the world: they identify shared, resp. class-specific traits, compositionality within the network, as well as locality in convolutional layers. Our rules are easily interpretable, but also super-charge prototyping as they identify which groups of neurons to consider in unison.

1 INTRODUCTION

Neural networks provide state of the art performance in many settings. How they perform their tasks, how they perceive the world, and especially, how the neurons within the network operate in concert, however, remains largely elusive. While there exists a plethora of methods for explaining neural networks, most focus on the mapping between input and output (e.g. model distillation) or only characterize individual neurons within the network (e.g. prototyping). In this paper, we introduce a new method that explains how the neurons in a neural network interact. In particular, we consider the activations of neurons in the network over a given dataset, and propose to characterize these in terms of rules $X \rightarrow Y$, where X and Y are sets of neurons in different layers of the network. A rule hence represents that neurons Y are typically active when neurons X are. For robustness we explicitly allow for noise, and to ensure that we discover succinct and non-redundant rules we formalize the problem in terms of the Minimum Description Length principle. To discover good rule sets, we propose the unsupervised EXPLAINN method. The rules that we discover give clear insight in how the network performs its task. As we will see, they identify what the network deems similar and different between classes, show how information flows within the network, and show which convolutional filters it expects to be active where. Our rules are easily interpretable, give insight in the differences between datasets, show the effects of fine-tuning, as well as super-charge prototyping as they tell which neurons to consider in unison.

Explaining neural nets is of widespread interest, and especially important with the emergence of applications in healthcare and autonomous driving. In the interest of space we here shortly introduce the work most relevant to ours, while for more information we refer to surveys (Adadi & Berrada, 2018; Ras et al., 2018; Xie et al., 2020; Gilpin et al., 2018). There exist several proposals for investigating how networks arrive at a decision for a given sample, with saliency mapping techniques for CNNs among the most prominent (Bach et al., 2015; Zhou et al., 2016; Sundararajan et al., 2017; Shrikumar et al., 2017). Although these provide insight on what parts of the image are used, they are inherently limited to single samples, and do not reveal structure across multiple samples or classes. For explaining the inner working of a CNN, research mostly focuses on feature visualization techniques (Olah et al., 2017) that produce visual representations of the information captured by neurons (Mordvintsev et al., 2015; Gatys et al., 2015). Although these visualizations provide insight on how CNNs perceive the world (M. Øygard, 2016; Olah et al., 2018) it has been shown that concepts are often encoded over multiple neurons, and that inspecting individual neurons does not

provide meaningful information about their role (Szegedy et al., 2013; Bau et al., 2017). How to find such groups of neurons and how the information is routed between layers remains unsolved.

An orthogonal approach is that of model distillation, where we train easy to interpret white box models that mimic the decisions of a neural network (Ribeiro et al., 2016; Frosst & Hinton, 2017; Tan et al., 2018). Rules of the form *(if–then)* are easily interpretable, and hence a popular technique for model distillation (Taha & Ghosh, 1999; Lakkaraju et al., 2017). Although some consider individual neurons to determine what rules to return (Robnik-Šikonja & Kononenko, 2008; Özbakır et al., 2010; Barakat & Diederich, 2005) all these techniques only yield rules that directly map input to output, and hence do not provide insight into how information flows through the network.

We instead propose to mine sets of rules to discover groups of neurons that act together across different layers in feed forward networks, and so reveal how information is composed and routed through the network to arrive at the output. To discover rules over neuron activations, we need an unsupervised approach. While a plethora of rule mining methods exists, either based on frequency (Agrawal & Srikant, 1994; Bayardo, 1998; Moerchen et al., 2011) or statistical testing (Hämäläinen, 2012; Webb, 2010), these typically return millions of rules even for small datasets, thus undermining the goal of interpretability. We therefore take a pattern set mining approach (Fischer & Vreeken, 2019), where we are after that set of rules that maximizes a global criterion, rather than treating each rule independently. The traditional pattern language of conjunctions is too restrictive for our setting, and hence we present EXPLAINN, a non-parametric and unsupervised method that discovers sets of rules that can model conjunctions, disjunctions, as well as approximate conjunctions.

2 THEORY

We first informally discuss how to discover association rules between neurons. We then formally introduce the concept of robust rules, and how to find them for arbitrary binary datasets, last, we show how to combine these ideas to reveal how neurons are orchestrated within feedforward networks.

2.1 PATTERNS OF NEURON CO-ACTIVATION

Similar to neurons in the brain, when they are active, artificial neurons also send information along their outgoing edges. To understand flow of information through the network, it is hence essential to understand the activation patterns of neurons between layers. Our key idea is to use recent advances in pattern mining to discover a succinct and non-redundant set of rules that together describe the activation patterns found for a given dataset. For two layers I_i, I_j, these rules $X \rightarrow Y, X \subset I_i, Y \subset I_j$ express that the set of neurons Y are usually co-activated when neurons X are co-activated. That is, such a rule provides us *local* information about co-activations within, as well as the dependence of neurons between layers. Starting from the output layer, we discover rules between consecutive layers I_j, I_{j-1}. Discovering overlapping rules between layers $X \rightarrow Y$ and $Y \rightarrow Z, X \subset I_j, Y \subset I_{j-1}, Z \subset I_{j-2}$, allows us to trace how information flows through the entire network.

Before we can mine rules between layers I_i and I_j of a network, we have to obtain its binarized activations for a given data set $\mathcal{D} = \{d_k = (s_k, o_k)\}$. In particular, for each sample s_k and layer I_i, we take the tensor of activations ϕ_i and binarize it to ϕ_i^b – in our experiments, we consider state of the art networks with *ReLU* activations, which binarize naturally at threshold 0. We gather these into a binary dataset D where each row t_k corresponds to the concatenation of ϕ_i^b and ϕ_j^b of I_i and I_j for s_k, i.e., $t_k \in D$ is a binary vector of length $|I_i| + |I_j|$. See Fig. 1 for a toy example.

Next, given D, our goal is to find that set of rules that together succinctly describe the observed activations. The Minimum Description Length (MDL) principle lends itself as an objective to find such sets. MDL is a statistically well-founded and computable approximation of Kolmogorov Complexity (Li & Vitányi, 1993). First introduced by Rissanen (1978), the essential idea is that the model $M^* \in \mathcal{M}$ that best describes data D is the model that losslessly encodes D using the fewest bits $M^* = \arg\min_{M \in \mathcal{M}} L(D, M)$. Here, our model class $\mathcal{M}$ is the superset of all possible rules over D, and by MDL we identify the best model M^* as the one that compresses the data best. Traditionally, rule mining is restricted to conjunctions over items, which is not sufficient for our application; neuron activations tend to be noisy, labels are inherently mutually exclusive, and hence we consider an extended language that allows for partial disjunctions of items (neurons, labels) and introduce a codelength function $L(D, M)$ to instantiate MDL for our model class of rule sets.

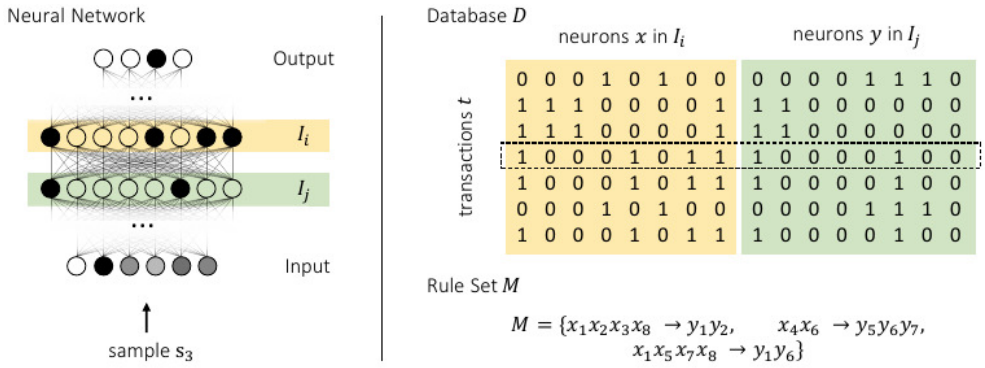


Figure 1: *Overview.* For a given network (left), binarized activations are gathered for the layers I_i, I_j for each sample, and summarized in the binary database D (right). Rules are discovered over D, where a good rule set M is given with at the bottom right, with rules $X \rightarrow Y$, $X \in I_i,\ Y \in I_j$.

2.2 MDL for Robust Rules

Our goal is to find a set of rules M that, in terms of description length $L(D, M)$, best describes a binary database $D = \{t \mid t \subset \mathcal{I}\}$ that consists of transactions t that are subsets of items $\mathcal{I}$. Each rule is of the form $X \rightarrow Y,\ X, Y \subset \mathcal{I}$, and indicates that Y is strongly associated with, i.e. occurs mostly in transactions where X is present. We say a rule $X \rightarrow Y$ *applies* to a transaction t iff $X \subset t$ and say a rule *holds* for t if additionally $Y \subset t$. We indicate these transactions sets as $T_X = \{i \mid t_i \in D.\, X \subset t_i\}$, respectively $T_{Y|X} = \{i \mid t_i \in T_X.\, Y \subset t_i\}$. Based on these definitions of rule transaction sets, we can now formally introduce our codelength function $L(D, M)$.

Baseline model Our base model $M_{ind} = \{\emptyset \rightarrow I \mid \forall I \in \mathcal{I}\}$ consists of singleton rules only, i.e. it models that all items $\mathcal{I}$ are generated independently. To send the n transactions of D using M_{ind}, we simply send for each item I in which out of all transactions in the database it appears. We can do so optimally using a log binomial code, which is given by $\log \binom{|T_\emptyset|}{|T_{I|\emptyset}|} = \log \binom{n}{|T_I|}$. To unambiguously decode, the recipient needs to know each $|T_I|$, which we can optimally encode via the parametric complexities of the binomials, which are defined as

$$L_{pc}(n) = \log \left(\sum_{k=0}^{n} \frac{n!}{(n-k)!k!} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k} \right),$$

and can be computed in linear time (Kontkanen & Myllymäki, 2007). We thus have $L(D, M_{ind}) = \sum_{I \in \mathcal{I}} \left(\log \binom{n}{|T_I|} + L_{pc}(n) \right)$. M_{ind} serves as our baseline model, and its singleton rules are a required part of any more complex model as they ensure we can always send any data over $\mathcal{I}$.

Non-trivial models A non-trivial model M contains rules of the form $X \rightarrow Y,\ \ X, Y \subset \mathcal{I}$ that are not part of M_{ind}. The idea is that we first transmit the data for where these non-trivial rules hold, and then send the remaining data using M_{ind}. To determine where such a rule applies, the receiver needs to know where X holds, and hence the data over X needs to be transmitted first. To ensure that we can decode the data, we only consider models M for which the directed graph $G = (\mathcal{I}, E)$ is acyclic, in which there exists an edge between two items i_1, i_2 iff they occur in the head and tail of a rule, that is $E = \{(i_1, i_2) \mid \exists X \rightarrow Y \in M.\, i_1 \in X \wedge i_2 \in Y\}$. We thus get a codelength

$$L(D \mid M \cup M_{ind}) = \left(\sum_{X \rightarrow Y \in M} \log \binom{|T_X|}{|T_{Y|X}|} \right) + \sum_{\emptyset \rightarrow I \in M_{ind}} \log \binom{n}{|T'_I|},$$

where $T'_I = \{t \in D \mid (I \in t) \wedge (\forall X \rightarrow Y \in M.\, I \in Y \implies t \notin T_{Y|X})\}$ is a modified transaction set containing all transactions with item I not covered by any non-trivial rule.

In addition to the parametric complexities of the binomial codes, the model cost of a non-trivial model also includes the cost of transmitting the non-trivial rules. To transmit a rule $X \rightarrow Y$, we

first send the cardinalities of X resp. Y using the universal code for integers $L_{\mathbb{N}}$ (Rissanen, 1983). For $n \geq 1$, this is defined as $L_{\mathbb{N}}(z) = \log^* z + \log c_0$ with $\log^*(z) = \log z + \log\log z + \ldots$, summing only over the positive components (Rissanen, 1983). To satisfy the Kraft equality we set $c_0 = 2.865064$. Knowing the cardinalities, we can then send the items of X resp. Y one by one using an optimal prefix code given by $L(X) = -\sum_{x \in X} \log \frac{|T_x|}{\sum_{I \in \mathcal{I}} |T_I|}$. For a particular rule $X \to Y \in M$, the model costs thus amount to

$$L(X \to Y) = L_{\mathbb{N}}(|X|) + L_{\mathbb{N}}(|Y|) + L(X) + L(Y) + L_{pc}(|T_X|) \ .$$

The overall model cost hence become the cost of the parametric complexities for the baseline rules plus the costs of the non-trivial rules

$$L(M \cup M_{ind}) = |\mathcal{I}| \times L_{pc}(n) + L_{\mathbb{N}}(|M|) + \sum_{X \to Y \in M} L(X \to Y) \ .$$

We provide an example in App. A.1. With these definitions, we have an MDL score that identifies the best rule set M^* for data D as

$$M^* = \underset{M \in \mathcal{M}}{\arg\min} \left(L(M \cup M_{ind}) + L(D \mid M \cup M_{ind}) \right) \ ,$$

where $\mathcal{M}$ is the space of all possible rule sets over the items in D.

Robust Rules In real world applications, which are inherently noisy, we need a more robust score that can approximate where rules hold and apply. In addition, as output neurons are only active mutually exclusively, rules need to be able to model disjunctions. The key problem with noisy data is that a single missing item in a transaction can cause a whole rule not to hold, or even apply. To discover rules that generalize well, we need to explicitly account for noise. The idea is to let rules apply, and hold, approximately. Specifying how many items l, and k, out of all items in the rule head, respectively tail, need to be part of a transaction, we relax the original rule definition to account for random noise (see App. Figs. 6, 7). Setting $l = 1$ and $k = 1$ coincidentally corresponds to a disjunction of items in the head and tail of the rule $X \to Y$, thus allowing to model output neurons correctly, and $l = |X|$ and $k = |Y|$ correspond to the original stringent rule definition. Varying between the two extremes accounts for varying levels of noise. The optimal l and k are those that minimize the MDL score. To ensure a lossless encoding, we need to make sure that the receiver can reconstruct the original data. Thus, for the relaxed head definition, for each rule we send the number of items l for the rule to apply using $L_{\mathbb{N}}(l)$ bits. Knowing each l, the receiver can reconstruct where each rule applies. Sending where a rule holds now leaves the receiver with an approximation of the data. To be able to reconstruct the actual data, Fischer & Vreeken (2019) introduced error matrices that when XORed with the approximation yield the original data. These two matrices $\mathcal{X}^+_{X \to Y}$, and $\mathcal{X}^-_{X \to Y}$ correct for the errors made in the part where the rule applies and holds, respectively applies but does not hold (for an example see App. Fig. 6). These error matrices are part of the model M and have to be transmitted with an adapted $L(D, M)$, which we review in App. A.2.

Complexity of the search To discover rules over the activations of two layers I_i, I_j, we have to explore the search space of all sets of rules, with a rule being formed by subsets of neurons in I_i for the head, combined with any subset of neurons of I_j for the tail. There exist $2^{|I_i|} \times 2^{|I_j|}$ such rules, and hence $2^{2^{|I_i|+|I_j|}}$ distinct models would need to be explored. Fischer & Vreeken (2019) showed that the rule set search space does not lend itself to efficient search as it is neither monotone nor submodular, the counterexamples also holding for our model definition. In fact, for robust rules, we additionally have to consider where rules should apply respectively hold – optimizing k and l – which results in approximately $2^{|I_i| \times |I_j| \times 2^{|I_i|+|I_j|}}$ models (detailed derivation in App. A.4). Exhaustive search is therewith infeasible, which is why we present EXPLAINN, a heuristic algorithm to efficiently discover good sets of rules.

2.3 DISCOVERING GOOD RULE SETS WITH EXPLAINN

EXPLAINN is based on the idea of iteratively refining the current model by merging and refining already selected rules. The key insight of the algorithm is that for a rule $X \to Y$ to summarize the

data well, also rules $X \rightarrow Y'$ with only part of the tail, $Y' \subset Y$, should summarize well, as all tail items should be similarly co-occuring with head X. Starting from the baseline model M_{ind} we iteratively and greedily search for better models until we can no longer improve the MDL score. As search steps we consider either introducing a new rule to M, by taking a good set of items $X \subset I_i$ for the head and a single item $A \in I_j$ for the tail and refine the model to $M' = M \oplus \{X \rightarrow A\}$, seeing if it decreases the overall MDL costs (Eq. 2.2). Or, we merge two existing rules $r_1 = X \rightarrow Y_1 \in M$ and $r_2 = X \rightarrow Y_2 \in M$, to form a new rule $r' = X \rightarrow Y_1 \cup Y_2$ and refine the model to $M' = M \oplus \{r'\} = (M \setminus \{r1, r2\}) \cup \{r'\}$. For a rule r', the refinement operator $\oplus$ is adding the rule $r' = X \rightarrow Y$ to M, removing the merged rules that led to r', if any, and updating the singleton transaction lists T_A for all items $A \in Y$, removing all transactions where r' holds.

To make this search scheme applicable to the scale of a typical neural net, we have to discuss how to efficiently search for candidate rules, with heads that can express anything from conjunctions to disjunctions. Immediately after, we present the full algorithm EXPLAINN for mining high quality rule sets for two arbitrary sets of neurons (e.g. layers) of a network.

Searching for candidates The key component in our iterative algorithm is the candidate generation process, which implements the two possible steps of generating new and merging existing rules. Given two layers I_i, I_j, to efficiently discover rules that are both robust to noise, and may include disjunctively active neurons in the head, we can not enumerate all possible rule heads for each individual tail neuron, as this would result in $|I_j| \times 2^{|I_i|}$ many rules. Instead, we keep a list H_y for each item $y \in I_j$, storing all head neurons $x \in I_i$ for which y is frequently active when x is active, that is $\sigma_{x,y} = \frac{|T_x \cap T_y|}{|T_x|} > \theta$, where θ is a confidence threshold. The lists are sorted decreasing on σ. We then search in each H_y for the rule with the highest gain going over all unions of first $t = 1 \dots |H_y|$ neurons in the list. We add the rule $X \rightarrow y$ with highest gain to the list of candidates (see App. Alg. 1). To compute the gain, we consider all possible values $k = 1 \dots |X|$ to determine for which transactions $T_X^k = \{t \in D \mid |X \cap t| \geq k\}$ the rule should robustly apply, where $k = 1$ corresponds to disjunction and $k = |X|$ to conjunction.

For an individual neuron y, such a rule would be optimal, but, our goal is to discover groups of neurons that act in concert. To this end we hence iteratively merge rules with *similar* heads – similar, rather than same, as this gives robustness both against noise in the data, as well as earlier merging decisions of the algorithm. For two rules $X_1 \rightarrow Y_1,\ X_2 \rightarrow Y_2$ with symmetric difference $X_1 \ominus X_2 = (X_1 \setminus X_2) \cup (X_2 \setminus X_1)$, we consider possible candidate rules $X_1 \cup X_2 \rightarrow Y_1 \cup Y_2$ and $X_1 \cap X_2 \rightarrow Y_1 \cup Y_2$, iff $|X_1 \ominus X_2| \leq \mu$ for some threshold $\mu \in \mathbb{N}$. For example, $\mu = 1$ corresponds to the case that one head has one label more than the other, all other labels are the same.

EXPLAINN Assembling the above pieces, we arrive at EXPLAINN, which given two sets of neurons I_i, I_j and a database of activations of these neurons, yields a heuristic approximation to the MDL optimal model M^*. By first introducing all relevant single neuron rules, it then proceeds by iteratively merging existing rules using the approach described above, until it can achieve no more gain. For efficiency, we separate the generation of the new rules from the merging of existing rules. In practice, this does not harm performance, as we allow merging of similar heads and can thus revert too greedy decisions introduced by new rules with too large or small heads. Furthermore, by observing that independent rules $X_1 \rightarrow Y_1,\ X_2 \rightarrow Y_2,\ Y_1 \cup Y_2 = \emptyset$ do not influence each others impact on the codelength, we can add all independent rules with the highest respective gain at once. We give the pseudocode as App. Alg. 3.

Complexity of EXPLAINN The generation of new rules results in time $O(n \times |I_j| \times |I_i|^3)$, by iterating over each neuron in I_j, and considering each subset of the most overlapping neurons in I_i, and considering each threshold $k = 1 \dots |I_i|$ for when the rule should apply, and the factor n from intersecting transaction lists T to compute the gain. We can have at most $|I_j|$ generated rules before considering rule merges, and in every iteration of merging we combine two rules, reducing the rule set size by 1. In each such step, we consider $|I_j|^2$ merges, for each of which we compute the gain considering noisy head and tail. We thus have a worst case runtime of $O(n \times |I_j|^4 \times |I_i|)$. As MDL ensures we consider succinct models that capture only the relevant structure in the data, EXPLAINN is in practice much faster and easily scales to several thousands of neurons.

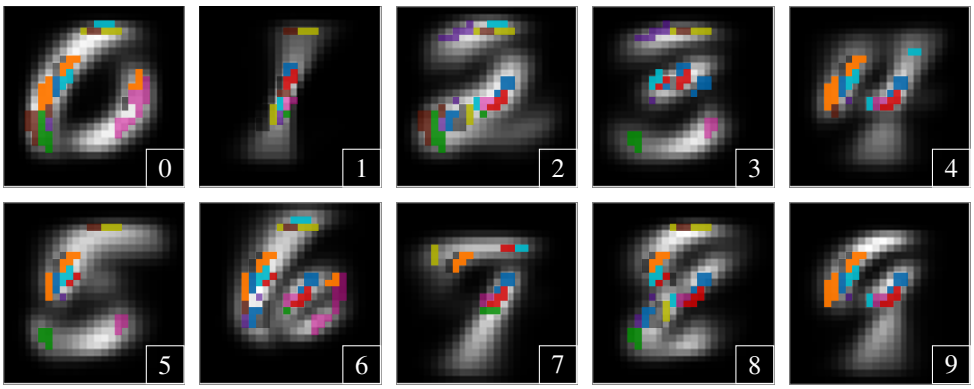


Figure 2: *MNIST* Average activation of neurons for the classes in *MNIST* for filter 2 in the first convolutional layer. Overlayed are the EXPLAINN rules, where pixel groups of the same colour (e.g. purple pixels top left for classes 2, 3) belong to a single rule spanning multiple classes.

3 EXPERIMENTS

In this section we evaluate how well the rule sets that EXPLAINN discovers provide insight in how CNNs perceive the world. We focus on CNNs as they count towards the most widespread use of feedforward networks and naturally lend themselves for visualization, which helps us to interpret the discovered rules. We consider CNNs of different complexity, including GoogLeNet and VGG-S. We first compare to traditional prototyping and activation map approaches on *MNIST* (LeCun & Cortes, 2010), and then examine which information is used how to arrive at classification for *ImageNet* (Russakovsky et al., 2014). Finally, we investigate the effect of fine-tuning in transfer learning on the Oxford *Flower* data (Nilsback & Zisserman, 2008). We will make the implementation of EXPLAINN publicly available upon publication. In the below experiments, on commodity hardware EXPLAINN needed up to minutes for *MNIST* and *Flower*, and up to 6 hours for *ImageNet*, to yield rule sets of several hundred up to 3000 rules, depending on the data set and choice of layers.

3.1 HOW IS INFORMATION FILTERED

As a first simple example we consider *MNIST*. We train a simple CNN of 2 convolutional and one fully connected layer using Keras, achieving 99% classification accuracy on test data (see App. B.1 for details). We are interested in what the individual filters learn about the digits, and how EXPLAINN reveals shared features across several classes. We compare to average activation maps and single neuron prototypes. Examining different filters, we observe that average class activation maps does not reveal the purpose of a filter, whereas the rules that EXPLAINN discovers identify which pixels together trigger a filter. As an example, consider Fig. 2, in which for filter 2 in layer 1 we visualize the discovered rules on top of the average activation maps (which just represents the original numbers). Whereas the rules identify shared structure (indicated by pixels of the same colour, in the same location, for different classes) such as top-left curve for 0, 4, and 9 (depicted in orange), the prototype (see App. Fig. 8) looks like a maze and does not reveal any insight. Similarly, for filter 36 in layer 2, we directly see from the discovered rules that this filter detects horizontal edges in a class specific manner, whereas prototyping and activation maps again fail to reveal this information (see App. Fig. 9). Interestingly, we also find filters that learn the negative of an image, with activated areas corresponding to the background of the digit (see App. Fig. 10).

3.2 HOW DOES INFORMATION FLOW

Next, we investigate a more complex setting and examine the activations for the *ImageNet* data set of pretrained VGG-S and GoogLeNet architectures (Chatfield et al., 2014; Szegedy et al., 2015). We here focus on VGG-S, as for this network there exist optimized prototyping methods that yield highly interpretable images for sets of neurons (M. Øygard, 2016). We provide the results for GoogLeNet in App. B.2.1. Starting with rules from the labels to the last layer, we observe that EXPLAINN yields

(a) Top: Malamute,
Bottom: Siberian Husky

`Siberian husky` AND NOT `Malamute`

`Siberian husky` AND `Malamute`

`Malamute` AND NOT `Siberian husky`

(b) Rule prototypes for FC7

Figure 3: *Neurons discriminating Huskies and Malamutes.* a) Huskies and Malamutes are very similar looking dog breeds. b) Prototypes for rules $X \rightarrow Y$ discovered for classes X, `Siberian husky` (red frame), class `Malamute` (yellow frame), resp. both (orange frame) and neurons Y in FC7. The neurons associated with both classes represent typical features shared between the two classes, those associated only with `Siberian huskies` show their slightly sharper, more defined head, while those associated only with `Malamutes` capture their more fluffy fur.

complex rules spanning multiple labels and multiple neurons in the last fully connected layer (FC7), which together encode the information shared between labels. Examples include the faces of certain dog breeds, as well as the coloured beaks of birds (see App. B.2.2). Notably, if we visualize these neurons individually, it is hard to extract anything meaningful from the images: the information is really encoded in the set of neurons that act together (see App. Fig. 13). We also observe cases where rules describe how the network discriminates between very similar classes. We give an example in Fig. 3 for the neurons EXPLAINN discovers to be associated with huskies, with malamutes, and for both classes together. These dog breeds are visually very similar, sharing a black–white fur pattern, as well as head and ear shapes. These traits are reflected by the neurons corresponding to the rule for both breeds. Looking closer, we can see that distinct traits, the more defined snout and ears of the husky, respectively the fluffy fur of the malamute, are picked up by the neuron sets discovered for the individual dog breeds. Beside discovering what shared and distinct traits the network has learned for classes, unlike with class prototypes we also find out when it learns differences *across* samples of the same class. For example, for the dog breed Great Danes, we discover not one but three rules that upon visualization each correspond to visually very different sub-breeds (see App. Fig. 14).

Next we investigate the information flow within the network. For each discovered rule $X \rightarrow Y$ from labels to FC7, we now apply EXPLAINN to discover rules from Y to the previous layer, FC6, and repeat this process up to the convolutional layers, to build a trace of rules that characterize how information is routed through the network before arriving at the final classification. We visualize one such trace in Fig. 4, which gives insight in how the network perceives totem poles. We find that the relevant neurons of FC7 and FC6 yield prototypes that show typical totem pole heads with animalistic faces. The neuron sets found for the different filters of the last convolutional layer CONV5, together detect specific parts of the class, including the vertical pole, and the rooftop-like structure, decorated with animalistic shapes with eyes, that is typically found at the top of a totem. We provide another example of an information trace in App. Fig. 15, which shows how the network learns to detect features located typically at certain parts of the image due to the very prevalent side-view photos taken from dogs, a clear sign of overfitting by the neural network.

3.3 WHAT IS THE EFFECT OF FINE TUNING

Finally, we show that EXPLAINN can elucidate the effect of fine-tuning in transfer learning. For this we consider Oxford *Flower* data Nilsback & Zisserman (2008), which consists of 8k images of flowers of 102 classes. For investigation, we consider both the vanilla VGG-S network trained on *ImageNet* from above, and a fine-tuned version from the Caffee model zoo[1]. We run EXPLAINN

[1]https://github.com/jimgoo/caffe-oxford102

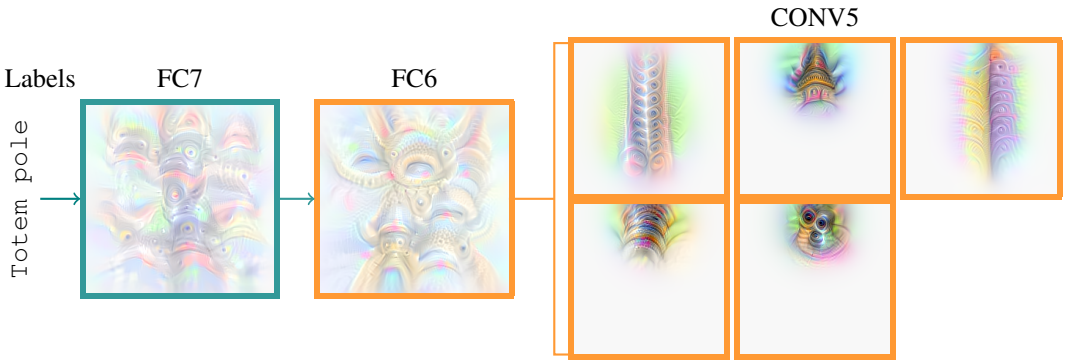


Figure 4: *Information flow for Totem poles.* Example from rule cascades found for *ImageNet* starting at label `Totem pole`. For each rule $X \rightarrow Y$, the group of neurons of the tail Y are used to generate a prototype. To discover rule cascades, we start mining rules between output and FC7 layer. Rule tails, which are neurons of FC7, are then used as heads for rules between FC7 and FC6. Similarly, rule tails are used as heads for rules between FC6 and CONV5.

to obtain rules between the output and the final layer of both networks, and give an example in App. Fig. 16. The visualizations show, as expected, a strong emphasize on colour and shape of the corresponding class of flowers. Interestingly, the visualizations of the same neurons for the original VGG-S network show almost identical shapes and pattern, but with less intense colour. Furthermore, we observe that some prototypes show animal-like features such as eyes or beaks.

4 Discussion and Conclusion

The experiments show that ExplaiNN is uniquely able to discover distinct groups of neurons that *together* capture traits as well as within-class heterogeneity. In particular, for convolutional filters, the rules directly aid in understanding how parts of the filters are used across classes to detect shared features, but also segment background, or detect edges. None of these aspects are revealed by prototyping, which can only consider an individual pixel or the whole filter, nor by activation maps, which miss the local information that patterns can provide.

We additionally discover distinct groups of neurons in fully connected layers that capture shared traits of classes, whereas other groups of neurons describe the distinct features of the respective classes, which helps in understanding how the network learns overarching features but still can discriminate between classes. Widely used approaches such as class prototypes, or visualization of individual neurons are not able to reveal similar information. Due to the local information that our rules provide, we can also detect differences the network sees across samples of a single class, where for example different groups of neurons describe visually different sub-breeds of a class of dogs. By connecting rules that we find across several layers, we can trace how information is gathered and combined to arrive at a classification, from filters that detect parts of the image that typically contain class specific features, through the fully connected layers where multiple neurons together encode the combined information, up to the final classification output.

Applying ExplaiNN to investigate the impact of fine-tuning in transfer learning, we found that for the groups of neurons discovered by our rules for the given fine-tuned CNN, surprisingly, the contained information is almost identical to the original CNN, but capturing the traits of the new classes almost perfectly. For the given transfer task the fine-tuning thus mostly resulted in routing information differently, rather than having to learn to detect and combine new features.

Overall, ExplaiNN performs well and convincingly answers 'what's in the box?'. It finds surprising results that help to understand how CNNs perceive the world. While many important tasks are solved by such networks, recurrent neural networks play an important role in e.g. language processing. Although rules can likely also help to understand what these models learn, these networks encode an entirely different type of information that is inherently hard to understand and visualize, and hence an exciting challenge for future work.

REFERENCES

Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile*, pp. 487–499, 1994.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

Nahla Barakat and Joachim Diederich. Eclectic rule-extraction from support vector machines. *International Journal of Computational Intelligence*, 2(1):59–62, 2005.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6541–6549, 2017.

Roberto Bayardo. Efficiently mining long patterns from databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Seattle, WA*, pp. 85–93, 1998.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *CoRR*, abs/1405.3531, 2014. URL `http://arxiv.org/abs/1405.3531`.

Jonas Fischer and Jilles Vreeken. Sets of robust rules, and how to find them. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Würzburg, Germany*. Springer, 2019.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pp. 80–89. IEEE, 2018.

Wilhelmiina Hämäläinen. Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures. *Knowledge and information systems*, 32(2):383–414, 2012.

Petri Kontkanen and Petri Myllymäki. A linear-time algorithm for computing the multinomial stochastic complexity. *Information Processing Letters*, 103(6):227–233, 2007. ISSN 0020-0190.

Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. Interpretable & explorable approximations of black box models. *arXiv preprint arXiv:1707.01154*, 2017.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, Heidelberg, 1993. ISBN 0387940537.

Audun M. Øygard. Peeking inside convnets. `https://www.auduno.com/2016/06/18/peeking-inside-convnets/`, June 2016.

Fabian Moerchen, Michael Thies, and Alfred Ultsch. Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression. *Knowledge and Information Systems*, 29(1):55–80, 2011.

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. 2015.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729. IEEE, 2008.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

Lale Özbakır, Adil Baykasoğlu, and Sinem Kulluk. A soft computing-based approach for integrated training and rule extraction from artificial neural networks: Difaconn-miner. *Applied Soft Computing*, 10(1):304–317, 2010.

Gabriëlle Ras, Marcel van Gerven, and Pim Haselager. Explanation methods in deep learning: Users, values, concerns and challenges. In *Explainable and Interpretable Models in Computer Vision and Machine Learning*, pp. 19–36. Springer, 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

Marko Robnik-Šikonja and Igor Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):589–600, 2008.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. URL `http://arxiv.org/abs/1409.0575`.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*, 2017.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Ismail A Taha and Joydeep Ghosh. Symbolic interpretation of artificial neural networks. *IEEE Transactions on knowledge and data engineering*, 11(3):448–463, 1999.

Sarah Tan, Rich Caruana, Giles Hooker, Paul Koch, and Albert Gordo. Learning global additive explanations for neural nets using model distillation. *arXiv preprint arXiv:1801.08640*, 2018.

Geoffrey I. Webb. Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM Transactions on Knowledge Discovery from Data*, 4(1):1–20, 2010.

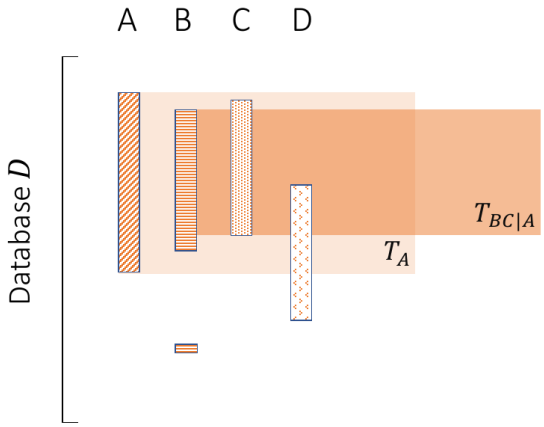
Ning Xie, Gabrielle Ras, Marcel van Gerven, and Derek Doran. Explainable deep learning: A field guide for the uninitiated. *arXiv preprint arXiv:2004.14545*, 2020.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

A MDL FOR ROBUST RULES

In this section we will give extended examples on how to compute the MDL score for a given database and set of rules, elaborate on the error encoding for the rule tails, and give a visual toy example on the impact of the extended pattern language for the rule head.

A.1 COMPUTING MDL FOR RULES



$$R_{ind} = \{\emptyset \to A, \emptyset \to B, \emptyset \to C, \emptyset \to D\}$$

$$R = R_{ind} \cup \{A \to BC\}$$

$$|D| = n = 100$$
$$|T_A| = 40$$
$$|T_B| = 35$$
$$|T_C| = 33$$
$$|T_D| = 35$$
$$|T_{BC|A}| = 30$$

Figure 5: *Example database and model.* A toy database D with blocks indicating where the items A, B, C, D occur in D, margins and relevant joint counts are given on the right. A sensible rule set $M \cup M_{ind} = A \to BC \cup M_{ind}$ is given on the right, the part of the database where the rule applies and holds is indicated by a light respectively dark orange area.

For the given example in Fig. 5, we will now compute the codelength $L(D, M) = L(M) + L(D \mid M)$ of transmitting the whole database D using $M \cup M_{ind}$. Here, we will stick with the simple encoding without error matrices, to make the process of computation more understandable. For reference, we first compute the baseline model, which is given by

$$\begin{aligned} L(D, M_{ind}) &= |\mathcal{I}| \times L_{pc}(|D|) + \sum_{I \in \mathcal{I}} \log \binom{|D|}{|T_I|} \\ &= 4 \times L_{pc}(100) + \log \binom{100}{40} + 2 \log \binom{100}{35} + \log \binom{100}{33} \\ &\approx 14.88 + 93.47 + 179.64 + 87.93 = \mathbf{375.92}. \end{aligned}$$

Thus, sending the data with just the baseline model costs 375.92 bits. Now, we will compute $L(D, M \cup M_{ind})$, we will start with the costs of sending the data $L(D \mid M \cup M_{ind})$

$$\begin{aligned} L(D \mid M \cup M_{ind}) &= \left(\sum_{X \to Y \in M} \log \binom{|T_X|}{|T_{Y|X}|} \right) + \left(\sum_{I \in \mathcal{I}} \log \binom{|D|}{|T'_I|} \right) \\ &= \log \binom{40}{30} + \log \binom{100}{40} + \log \binom{100}{5} + \log \binom{100}{3} + \log \binom{100}{35} \\ &\approx 29.66 + 93.47 + 26.17 + 17.30 + 89.82 = \mathbf{256.42}. \end{aligned}$$

The model costs are composed of the parametric complexities for the (adapted) baseline rules, plus the costs of transmitting what the rule is composed of along with its parametric complexity. We thus get

$$\begin{aligned} L(M \cup M_{ind}) &= |\mathcal{I}| \times L_{pc}(|D|) + \sum_{X \to Y \in M} L_{\mathbb{N}}(|X|) + L_{\mathbb{N}}(|Y|) + L(X) + L(Y) + L_{pc}(T_X) \\ &= 4 \times L_{pc}(100) + L_{\mathbb{N}}(1) + L_{\mathbb{N}}(2) - \log \frac{40}{143} - \log \frac{35}{143} - \log \frac{33}{143} + L_{pc}(40) \\ &\approx 14.88 + 1.52 + 2.52 + 1.84 + 2.03 + 2.12 + 3.11 = \mathbf{28.02}. \end{aligned}$$

Hence, the model with the complex rule has a smaller codelengh than the baseline, with $L(D, M \cup M_{ind}) = \mathbf{284.44}$ bits.

A.2 THE ERROR ENCODING FOR TAILS

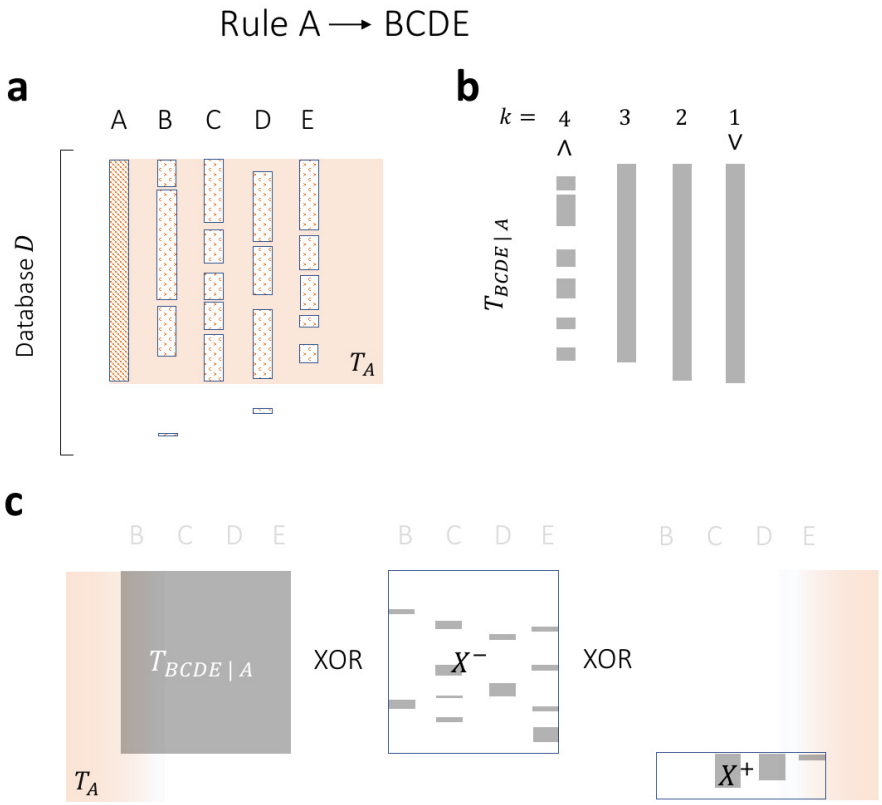


Figure 6: *Example of tail error encoding.* For a given database D given in **a**, where blocks indicate the occurrence of items, a good rule is given by$A \rightarrow BCDE$. The part of the database where the rule applies is indicated by the orange area. In **b** we show the part of the transaction were the rule holds for varying number k of tail items that have to be present in a transaction, from all items on the left – corresponding to a conjunction – towards just a single item on the right, which corresponds to a disjunction. In **c** we visualize the error encoding used to transmit the data for $k = 3$. We first transmit the data where the rule holds, resulting in the area that is indicated by the gray block. XORing the error matrix X^- with this block, it is possible to reconstruct the original data for the part where the rule holds. Using X^+, we reconstruct the original data in the area where the rule applies but does not hold.

For the error encoding for tails, which allow to discover rules in noisy settings (compare Fig. 6a,b), we send where a rule $X \rightarrow Y$ approximately holds according to some parameter k, which defines the number of items of the tail that have to be present in the transaction. The errors made by this approximation are then accounted for by sending error correcting matrices $\mathcal{X}^-_{X \rightarrow Y}$ and $\mathcal{X}^+_{X \rightarrow Y}$, which account for the destructive, respectively additive noise in the are where the rule applies (compare Fig. 6c).

Let us first assume we are given a k, we will later show how we can optimize for k. We redefine the transaction sets $T_{Y|X} = \{t \in D \mid (X \subset t) \wedge (|Y \cap t| \geq k)\}$, which corresponds to the transactions where the rule approximately holds. We will now slightly abuse notation and indicate the binary input matrix that correspond to D by $\mathcal{D}$, and we subset this matrix using the transaction id lists and item subsets. Both of these are sets of indices that indicate which rows, respectively columns to use of the matrix. For example, the submatrix where X holds is given by $\mathcal{D}[T_X, X]$.

We can now define the error correcting matrices to be $\mathcal{X}^-_{X\to Y} = \mathcal{D}[T_{Y|X}, Y] \otimes \mathbb{1}^{|T_{Y|X}|\times|Y|}$, and $\mathcal{X}^-_{X\to Y} = \mathcal{D}[T_X \setminus T_{Y|X}, Y]$, where $\otimes$ is the element-wise XOR operator and $\mathbb{1}^{i\times j}$ is a matrix of size $i \times j$ filled with ones. The receiver, knowing T_X and $T_{Y|X}$, can then reconstruct the original data $\mathcal{D}[T_{Y|X}, Y] = \mathbb{1}^{|T_{Y|X}|\times|Y|} \otimes \mathcal{X}^-_{X\to Y}$, respectively $\mathcal{D}[T_X \setminus T_{Y|X}, Y] = \mathcal{X}^+_{X\to Y}$.

While this explains the concept of how error correcting matrices can be used to reconstruct the original input, which hence define a lossless encoding, we are mainly interested in the codelength functions. To adapt the data costs, we now additionally send the two error matrices, which we can do using binomial codes. Hence, we get

$$L(D \mid M) = \left(\sum_{X\to Y\in M} \log\binom{|T_X|}{|T_{Y|X}|}\right) + \left(\sum_{I\in\mathcal{I}} \log\binom{|D|}{|T'_I|}\right) + \log\binom{|T_{Y|X}|\times|Y|}{|\mathcal{X}^-_{X\to Y}|} + \log\binom{|T_X \setminus T_{Y|X}|\times|Y|}{|\mathcal{X}^+_{X\to Y}|},$$

with the second line providing the codelength of the error matrices, and $|\mathcal{X}|$ indicating the number of ones in $\mathcal{X}$.

Our model M now not only consists of rules $M \cup M_{ind}$, but also of the set of error correcting matrices. As the submatrix to which we need to apply the matrix is fully defined by $T_X, T_{Y|X}$, and Y of the corresponding rule, also defining its size, the only adaptation we need for the model costs is the parametric complexities induced by the codes for transmitting the data. This yields

$$L(M) = |\mathcal{I}| \times L_{pc}(|D|) + \sum_{X\to Y\in M} L(X \to Y) + L_{pc}(|T_{Y|X}| \times |Y|) + L_{pc}(|T_X \setminus T_{Y|X}| \times |Y|).$$

This completes the MDL costs for rules robust to noise in the tail for a given k. To optimize k, the crucial insight is that the codelength of individual complex rules are independent, as is the data cost. That means we can optimize a k for each rule separately. Thus, for a given rule $X \to Y$ we can enumerate all $|Y|$ many models for the different thresholds k and let MDL decide which one fits the data best.

A.3 The impact of the extended pattern language

Extending the pattern language for rule heads is crucial to be applicable for tracing activation patterns through a neural network. First of all, we need to start from labels, which are inherently activated mutually exclusive, as we only have a single label as classification. To find shared features of labels, it is essential to be able to express disjunctions with rule heads. Furthermore, the data as well as activation patterns across the data are very noisy. Thus, determining where a rule applies just based on conjunctions of features can give a very twisted look of the data at hand, as visualized in Fig. 7. That is the reason to introduce a more flexible language similar to approximate rule tails, which solves these issues.

A.4 Search complexity

The size of the search space implied by our model class $\mathcal{M}$ is in $O(2^{|I_i|\times|I_j|\times 2^{|I_i|+|I_j|}})$. For two layers I_i, I_j, we enumerate all possible rules by

$$\underbrace{\Big(\sum_{k=0}^{|I_i|} k \times \binom{|I_i|}{k}\Big)}_{\text{Possibilities for head}} \times \underbrace{\Big(\sum_{l=0}^{|I_j|} l \times \binom{|I_j|}{l}\Big)}_{\text{Possibilities for tail}}$$
$$\leq |I_i|\Big(\sum_{k=0}^{|I_i|} \binom{|I_i|}{k}\Big) \times |I_j|\Big(\sum_{l=0}^{|I_j|} \binom{|I_j|}{l}\Big)$$
$$= |I_i|\, 2^{|I_i|} \times |I_j|\, 2^{|I_j|} = |I_j|\, |I_i|\, 2^{|I_i|+|I_j|},$$

where the first sum enumerates all heads of size k, the binomial coefficient describes the ways of drawing heads of such size, and the term k is the number of models given by the robust head

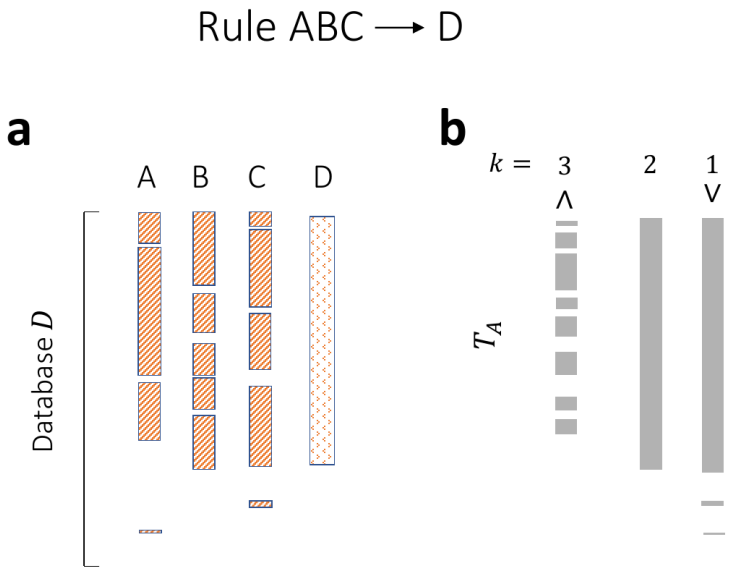


Figure 7: *Example of the impact of noise.* For a given database D given in **a**, where blocks indicate the occurrence of items, a good rule is given by$ABC \rightarrow D$. Due to high noise, the simple conjunctive pattern language results in a bad representation on where the rule should apply, visualized on the left of **b**. More relaxed definitions towards disjunctions, where we only require l items of the head to be present in the transaction, result in much more stable representation on where the rule applies.

encoding. Similarly, the second sum enumerates all tails of size l, the binomial coefficient describes the drawing of such tails, and the term l is the number of ways to place the error correcting matrices for the robust tail encoding. As in theory we can have any subset of these rules as a model, we thus get approximately $2^{(|I_j| \times |I_i| \times 2^{|I_i|+|I_j|})}$ many different models.

A.5 Algorithm Pseudocode

ExplaiNN explores the search space of rule sets in an iterative fashion, either generating new rules with a single item in the tail, or merging two existing rules, thus generating more complex rules with multiple items in the tail. Using these two steps, we can generate all potential candidate rules to add to the model, and evaluate their respective gain in terms of MDL. For a rule r', we will say model $M' = M \oplus r'$ is the refined model, with the refinement operator $\oplus$ adding the rule $r' = X \rightarrow Y$ to M, removing the merged rules that led to r', if any, and updating the singleton transaction lists T_A for all items in the tail $A \in Y$. Here, we will provide the pseudocode for the two candidate generation functions for new rules and for merging rules in the general setting alongside the complete algorithm of ExplaiNN.

For generating a new rule with a head using the extended pattern language we use the approach described in the main paper, gathering all confidence values for a given neuron A in I_j for all potential head neurons I_i. We keep all potential head neurons with confidence value beyond θ in a list H_A sorted descending on confidence and merge the first t neurons in the list to form the head. Going over all $t = 1..|H_A|$ allows us to greedily optimize for the best of all relevant heads for the given item,.

Algorithm 1: GENCANDIDATENEW

Input: Dataset D over layers I_i, I_j, Model M, tail item A, threshold θ
Output: Best refinement M'

1 $H_A \leftarrow \emptyset$; // head items, in decreasing order of confidence
2 **foreach** $x \in I_i$ **do**
3 $\quad \sigma_{x,A} \leftarrow \frac{|T_x \cap T_A|}{|T_x|}$; // Compute conditional frequency
4 $\quad$ **if** $\sigma_{x,A} > \theta$ **then**
5 $\quad\quad$ **insert** $(x, \sigma_{x,A})$ **into** H_A; // Add neuron x to list
6 $M' \leftarrow$ **none**;
7 $\Delta_{min} \leftarrow 0$; // gain estimate in bits
8 **for** $t = 1 ... |H_A|$ **do**
9 $\quad M' = M \oplus \{H_A[: t] \rightarrow A\}$; // Refine model with rule using first t labels
10 $\quad \Delta_t \leftarrow L(D, M) - L(D, M')$;
11 $\quad$ **if** $\Delta_t < \Delta_{min}$ **then**
12 $\quad\quad \Delta_{min} \leftarrow \Delta_t$;
13 $\quad\quad M' \leftarrow M \oplus \{H_A[: t] \rightarrow A\}$; // Update best rule set
14 **return** M'

The key component is hidden in the gain estimate in line 10, which for the given rule $X \rightarrow A$ determines the best value k of items in the head needed for a rule to apply. That is, we test all for all transactions sets determining where the rule applies $T_X^k = \{t \in D \mid |X \cup t| \geq k\}$ which one gives the best gain. To generate new rules going from the output layer to a hidden layer, we want to mine rules with disjunctive heads, which means we only have to consider T_X^1 – corresponding to a disjunction – in the search process.

To generate candidates from existing rules in M, we use an extended search scheme that allows to merge pairs of rules with approximately equal heads, having up to μ dissimilar items, measured by the symmetric set differences between $\ominus$.

Algorithm 2: GENCANDIDATEMERGES

Input: Dataset D, Model M, overlap threshold μ
Output: Candidates C sorted by gain Δ

$C \leftarrow \emptyset$; // Candidate rule merges
for $r_1 = X_1 \rightarrow Y_1 \in M, r_2 = X_2 \rightarrow Y_2 \in M$ **do**
$\quad$ **if** $|X_1 \ominus X_2| \leq \mu$ **then**
$\quad\quad \Delta_\cap \leftarrow L(D, M \oplus \{X_1 \cap X_2 \rightarrow Y_1 \cup Y_2\}) - L(D, M)$; // Gain of adding conjunction of heads
$\quad\quad$ **if** $\Delta_\cap < 0$ **then**
$\quad\quad\quad$ **insert** $(X_1 \cap X_2 \rightarrow Y_1 \cup Y_2,\ \Delta_\cap)$ **into** C ; // Add to candidates
$\quad\quad \Delta_\cup \leftarrow L(D, M \oplus \{X_1 \cup X_2 \rightarrow Y_1 \cup Y_2\}) - L(D, M)$; // Gain of adding disjunction of heads
$\quad\quad$ **if** $\Delta_\cup < 0$ **then**
$\quad\quad\quad$ **insert** $(X_1 \cup X_2 \rightarrow Y_1 \cup Y_2,\ \Delta_\cup)$ **into** C; // Add to candidates
return C

Using the candidate generation methods, we can now write down the pseudocode for EXPLAINN, which iteratively generates candidates and commits to the candidate with highest gain, until there is no more candidate that yields any gain in terms of MDL.

Algorithm 3: EXPLAINN

Input: Dataset D over layers I_i, I_j, frequency threshold θ, overlap threshold μ
Output: Best model M^*

```
M ← {∅ → A | A ∈ I_j} ;                                  // Initialize model with baseline rule set
foreach A ∈ I_j do
    R' ← GenCandidateNew(D, M, A, θ);                    // App. Alg. 1
    M' ← M ⊕ R';
    if L(D, M') < L(D, M) then
        M ← M';
repeat
    M̂ ← M;
    C ← GenCandidateMerges(D, M, μ);                     // App. Alg. 2
    𝒴 ← ∅;                                               // Keep track of independence of merged rules
    foreach X → Y ∈ C. Y ⊄ 𝒴 do
        M' ← M ⊕ {X → Y};                                // Refine model, test gain
        if L(D, M') < L(D, M) then
            M̂ ← M'; 𝒴 ← Y;
until M = M̂;
return M
```

B EXPERIMENTS AND DATA

Here, we detail the setup and training of the individual networks, and provide further experimental results.

B.1 MNIST TRAINING

We trained a CNN on the MNIST data set using the Keras framework, using 60000 images for training and 10000 images as hold out test set for evaluation. The network consists of 2 convolutional layers, with 20 filters in the first layer and 40 filters in the second layer, each using 3x3 kernels and 2x2 maxpooling. The convolutional layers are followed by a Dropout layer with dropout rate $.25$, and the flattened outputs are passed on to a fully connected layer with 64 nodes with *ReLU* activations. Then follows a dropout layer with rate $.5$ and the output layer of size 10 with softmax activations. The network was trained using AdaDelta with default parameters based on categorical cross entropy loss over 12 epochs using a batch size of 128. We gathered binarized activations across all filters and applied EXPLAINN to build rules from the output layer to the first respectively second convolutional layer.

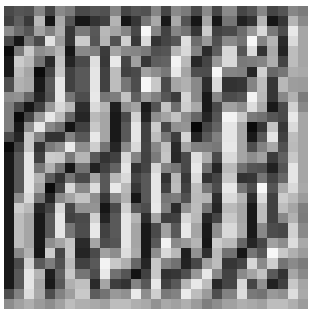
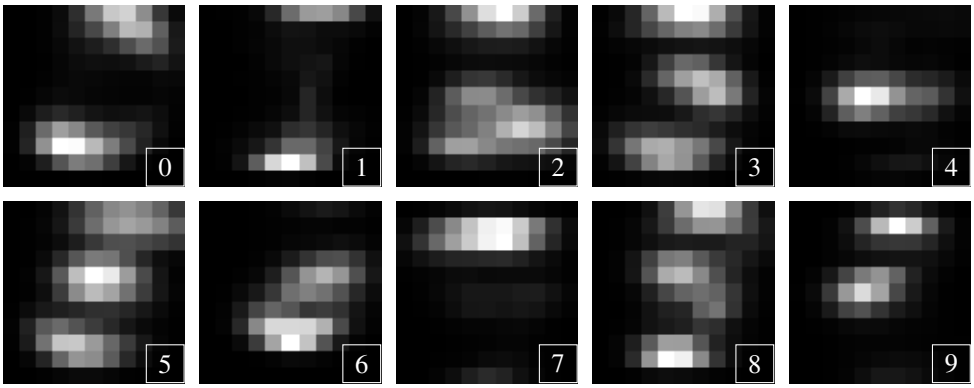
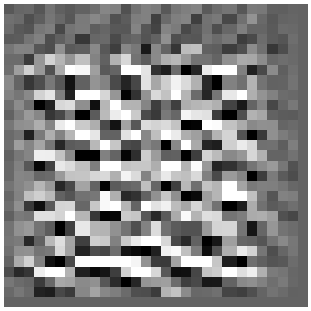


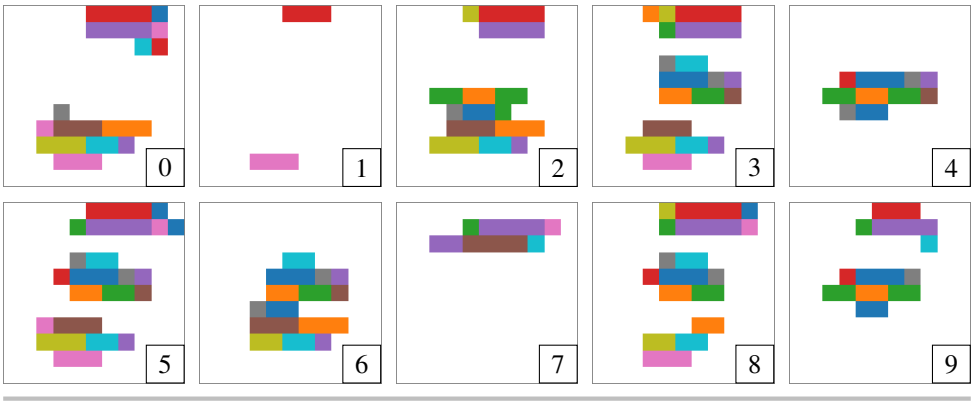
Figure 8: *MNIST prototype.* Prototype image for filter 2 from the first convolutional layer.



(a) *MNIST Average activations.* Average activation maps across a class for filter 32.



(b) *MNIST Prototype.* Prototype image for filter 32.



(c) *Horizontal edge detector.* Discovered rules, feature groups found across classes share the same colour.

Figure 9: *Filter visualizations.* Activation maps (a) for the classes, the prototype of the filter (b), and discovered rules (c), over the whole dataset for filter 36 in the second convolutional layer.

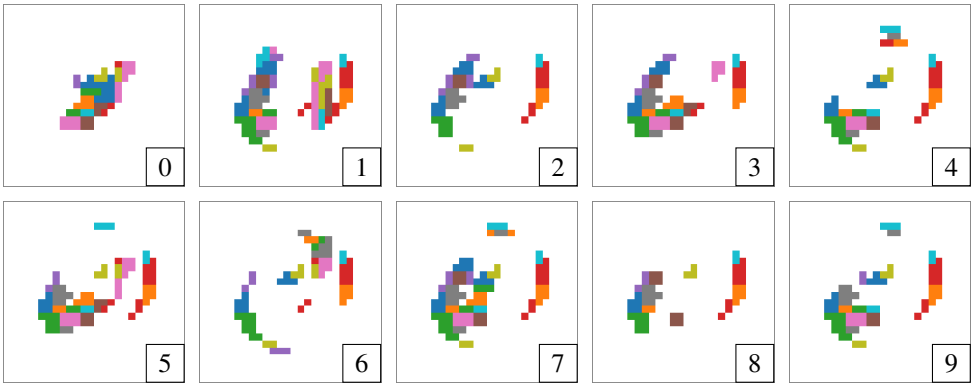
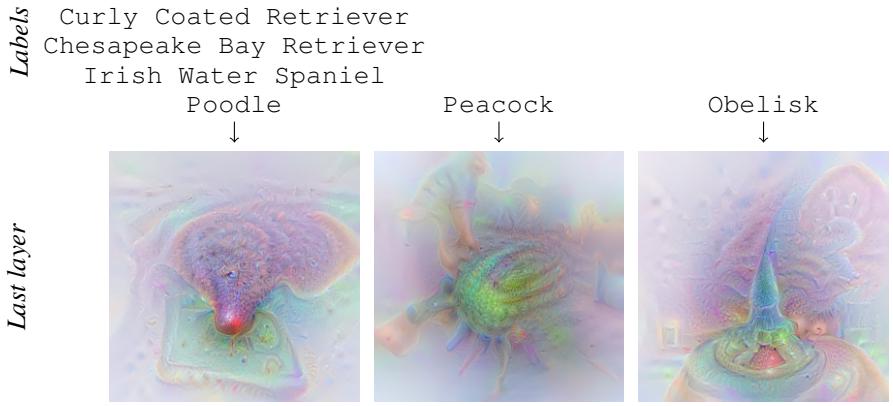


Figure 10: *The negative of a digit.* Visualizations for filter 12 in the first convolutional layer. This filter seems to capture the 'negatives' of the handwritten digits.

B.2 IMAGENET FURTHER RESULTS

B.2.1 GOOGLENET RESULTS



(a) Prototypes for rules from output label to last hidden layer.

(b) Samples for curly haired dog breeds. From left: `Curly Coated Retriever`, `Chesapeake Bay Retriever`, `Irish Water Spaniel`, `Poodle`

Figure 11: *GoogLeNet results on ImageNet.* (a)Visualizations for the rules found between the labels and the last hidden layer in GoogLeNet. The labels in the rule heads are written above the prototype images of the tail unit groups. Each rule tail captures some interesting features of the corresponding classes: In the first rule the characteristic curly hair of different dog breeds is captured, the second group encapsulates information about the typical colourful plumage of peacocks, the third captures the shape of obelisks. We provide example images of the curly haired dog breeds in (b).

B.2.2 VGG SHARED NEURONS

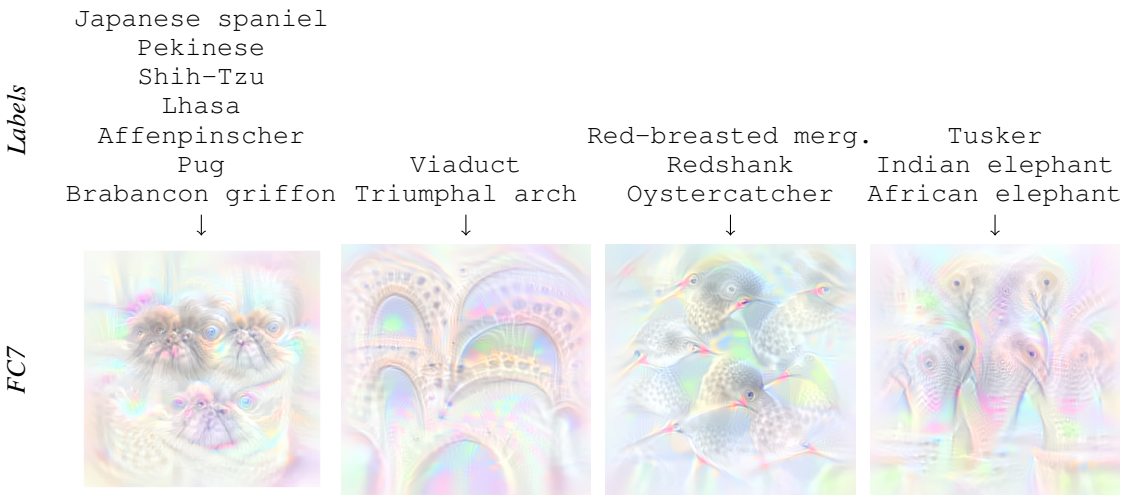
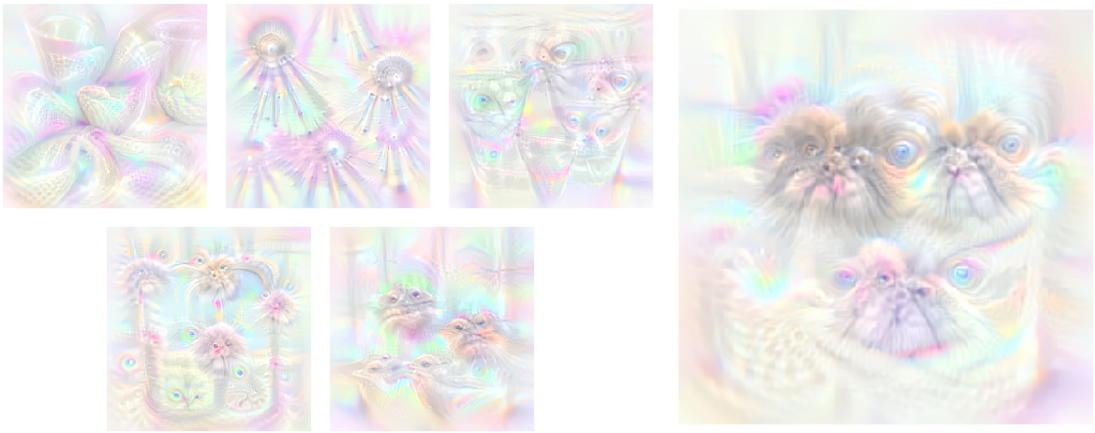


Figure 12: *Shared information across labels.* Visualizations for the rules found between the labels and the last fully connected layer (FC7). The labels in the rule heads are written above the prototype images of the tail unit groups. Each rule tail captures some interesting features of the corresponding classes: In the first rule the characteristic face of different dog breeds is captured, the second group encodes information about the arch structures present for both `Viaduct` and `Triumphal arch`, the third captures the red beaks surrounded by blackish feather that are shared between different birds, and the fourth shows typical heads and tusks of elephants.



Visualization for the units in the tail individually

Visualization for the whole tail

Figure 13: *The ugly face neurons.* From the data for all dog breed categories, EXPLAINN discovered the rule between the labels {`Japanese spaniel, Pekinese, Shih-Tzu, Lhasa, Affenpinscher, Pug, Brabancon griffon`}, and 5 units from the *FC7* layer, for which a prototype is given in the right image. The 5 units together capture the rather characteristic face of the mentioned breeds. Visualizing these 5 units one by one, given on the left, gives only little insight about the encapsulated information

We discovered many shared traits that the networks is able to pick up across classes, which are encapsulated in groups of neurons in the last layer. For example, there are neurons that capture the red beaks of different birds, arch like structures of buildings, tusks of elephants and as an object itself, and the ugly face of a whole group of different dog breeds (see App. Fig. 12). Even if we would visualize all neurons individually and handpick them, without the knowledge about the rule,

e.g. for the ugly dogs, it is very hard to interpret what they encode, and only few of them give hints about what the information is (see App. Fig. 13). In practice, we only visualize class prototypes, which can be very misleading, as shown in the next section.

B.2.3 RESOLVING THE MEANING OF CLASS PROTOTYPES

Class prototype for `Great Danes`

Row 1: Example images from the class
Row 2: Rules in association with `Great Danes`

Figure 14: The left image shows the visualization for the whole class `Great Danes`. This visualization could not highlight many characteristic features, since there is a large diversity within the class. On the right side 3 images from the dataset is shown, along with 3 rules that EXPLAINN finds in connection with the class label. We are able to pick up trends, that are not characteristic to the whole class, but only a subset.

A standard technique to capture traits that a neural network learns about a class is prototyping the given class label. These can offer hints on what the network learns globally about the class, but very often lead to uninterpretable results. We provide one such example prototype for the `Great Dane` class in Imagenet of the VGG network in App. Fig. 14, which does not provide any clue what the network learns. The rules discovered by EXPLAINN however show, that different groups of neurons in the last layer lead to the `Great Dane` classification, each encoding a distinct type of fur colour and pattern that appear with this breed. The class prototype is a mixture of these different types, which explains the difficulty to interpret that prototype.

B.3 OXFORD FLOWER DATA

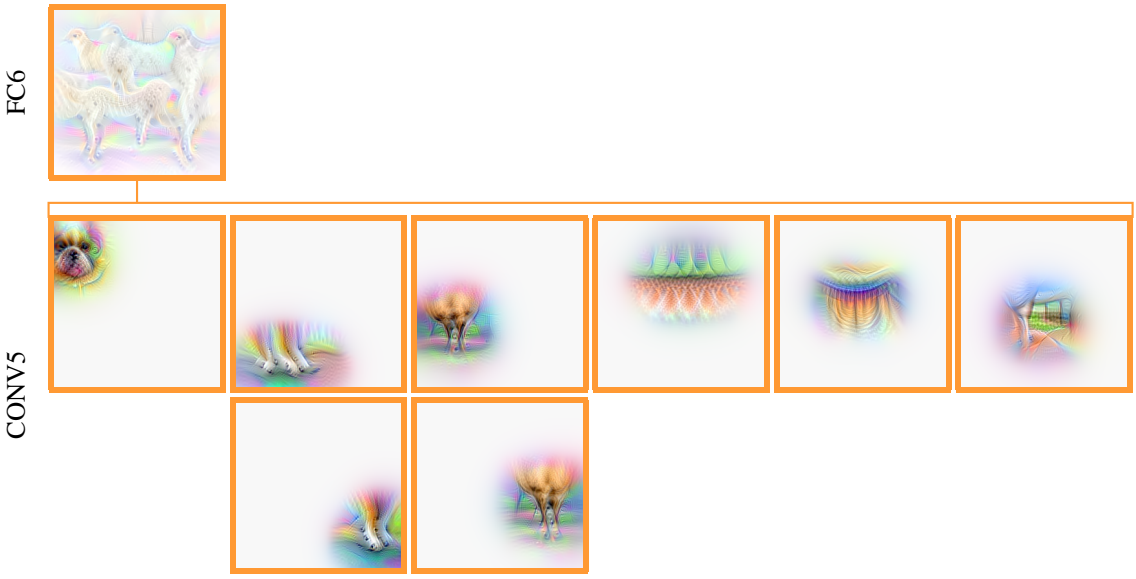


Figure 15: *Side dogs*. The visualization for the *FC6* nodes resembles animals from side view, a typical image positioning for dogs. In the images generated for associated units in the convolutional layer, we see how the network uses different filters to put this pattern together. The first depicted filter captures a dog head, typically found at the top left part of the image. The second and third filters focus on the legs. In the third row, we can see that other units for these filters are also commonly activated with the feature, showing that there are typical positions for legs in both the left and right lower parts of the image. In the last three prototypes, we recognize horizontal lines, which overall resembles the typical shape and position in the image where we would expect the back of an animal.

Figure 16: *Flower visualizations.* For rules found between output and last fully connected layer, we visualize the neurons in the tail of the rule for the fine-tuned VGG-S network (first), the original VGG-S network (second), and example images for the flower classes (right).