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

For the given example in Fig. 6, 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

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

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

A.2. The error encoding for tails

For the error encoding for tails, which allow to discover rules in noisy settings (compare Fig. 7a,b), we send where a rule $X \to Y$ 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 $\mathcal{X}^-_{X \to Y}$ and $\mathcal{X}^+_{X \to Y}$, which account for the destructive, respectively additive noise in the are where the rule applies (compare Fig. 7c).

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

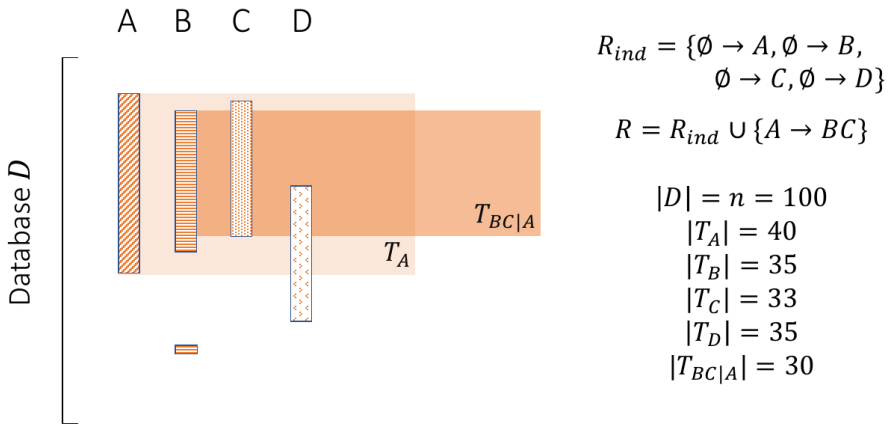


Figure 6: *Example database and model.* A toy database D with blocks indicating where the items A, B, C, D occur in D, margins and relevant joint counts are given on the right. A sensible rule set $M \cup M_{ind} = A \to BC \cup M_{ind}$ is given on the right, the part of the database where the rule applies and holds is indicated by a light respectively dark orange area.

additionally send the two error matrices, which we can do using binomial codes. Hence, we get

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

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

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

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. 8. 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

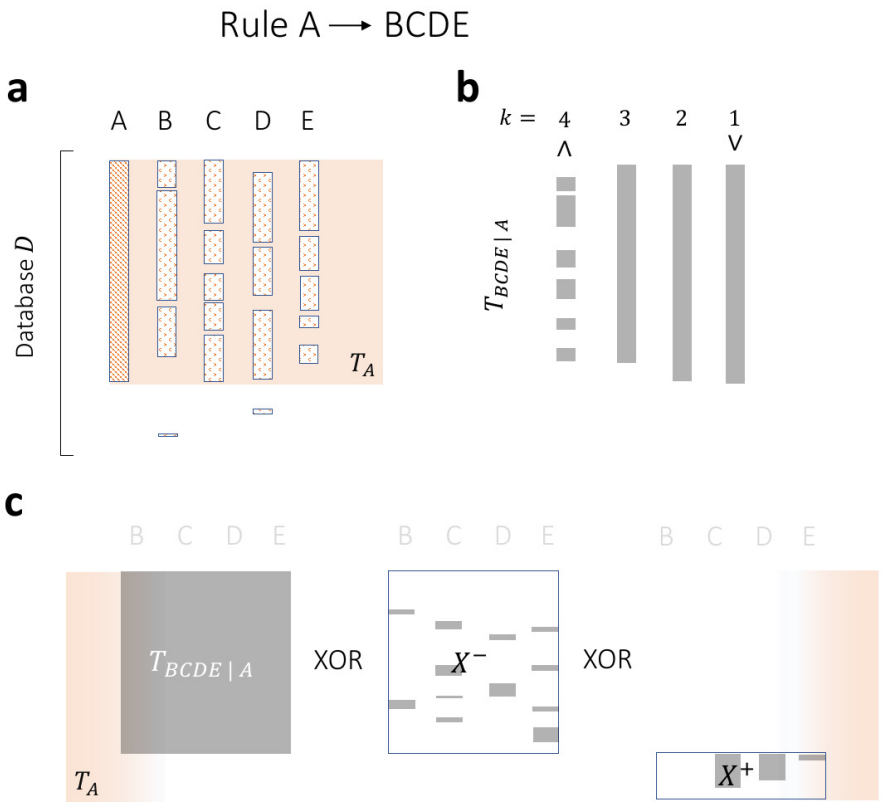


Figure 7: *Example of tail error encoding.* For a given database D given in **a**, where blocks indicate the occurrence of items, a good rule is given by$A \to BCDE$. The part of the database where the rule applies is indicated by the orange area. In **b** we show the part of the transaction were the 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.

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 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 $r' = X \to Y$ 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

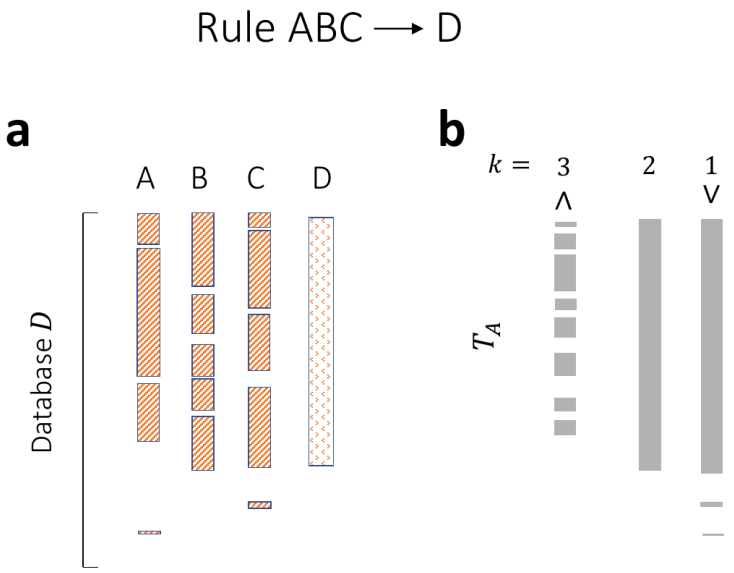


Figure 8: *Example of the impact of noise.* For a given database D given in **a**, where blocks indicate the occurrence of items, a good rule is given by$ABC \to D$. Due to high noise, the simple conjunctive pattern language results in a bad representation on where the rule should apply, visualized on 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.

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. We give pseudocode for generating new candidate rules in Alg. 1.

The key component is hidden in the gain estimate in line 10, which for the given rule $X \to A$ 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 $\ominus$. We provide pseudocode for this process in Alg. 2.

Algorithm 1 GenCandNew

Input: dataset D over layers I_i, I_j, Model M, tail item A, threshold θ
Output: best refinement M'
$H_A \leftarrow \emptyset$ {head items, in decreasing order of confidence}
for $x \in I_i$ **do**
  $\sigma_{x,A} \leftarrow \frac{|T_x \cap T_A|}{|T_x|}$ {Compute conditional frequency}
  if $\sigma_{x,A} > \theta$ **then**
    insert $(x, \sigma_{x,A})$ **into** H_A {Add neuron x to list}
  end if
end for
$M' \leftarrow \emptyset$
$\Delta_{min} \leftarrow 0$ {gain estimate in bits}
for $t = 1... |H_A|$ **do**
  $M' = M \oplus \{H_A[:t] \to A\}$ {Refine model with rule using first t labels}
  $\Delta_t \leftarrow L(D, M) - L(D, M')$
  if $\Delta_t < \Delta_{min}$ **then**
    $\Delta_{min} \leftarrow \Delta_t$
    $M' \leftarrow M \oplus \{H_A[:t] \to A\}$ {Update best rule set}
  end if
end for
return M'

Using the candidate generation methods, we can now write down EXPLAINN as given in Alg. 3, which iteratively

Algorithm 2 GenCandMerges

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**
  if $|X_1 \ominus X_2| \leq \mu$ **then**
    $\Delta_\cap \leftarrow L(D, M \oplus \{X_1 \cap X_2 \rightarrow Y_1 \cup Y_2\}) - L(D, M)$ {Gain of adding conjunction of heads}
    if $\Delta_\cap < 0$ **then**
      insert $(X_1 \cap X_2 \rightarrow Y_1 \cup Y_2,\ \Delta_\cap)$ **into** C {Add to candidates}
    end if
    $\Delta_\cup \leftarrow L(D, M \oplus \{X_1 \cup X_2 \rightarrow Y_1 \cup Y_2\}) - L(D, M)$ {Gain of adding disjunction of heads}
    if $\Delta_\cup < 0$ **then**
      insert $(X_1 \cup X_2 \rightarrow Y_1 \cup Y_2,\ \Delta_\cup)$ **into** C {Add to candidates}
    end if
  end if
end for
return C

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^*
$M \leftarrow \{\emptyset \rightarrow A \mid A \in I_j\}$ {Initialize model with baseline rule set}
for $A \in I_j$ **do**
  $R' \leftarrow$ GENCANDNEW(D, M, A, θ) {App. Alg. 1}
  $M' \leftarrow M \oplus R'$
  if $L(D, M') < L(D, M)$ **then**
    $M \leftarrow M'$
  end if
end for
repeat
  $\hat{M} \leftarrow M$
  $C \leftarrow$ GENCANDMERGES(D, M, μ) {App. Alg. 2}
  $\mathcal{Y} \leftarrow \emptyset$ {Keep track of independence of merged rules}
  for $X \rightarrow Y \in C.\ Y \not\subset \mathcal{Y}$ **do**
    $M' \leftarrow M \oplus \{X \rightarrow Y\}$ {Refine model, test gain}
    if $L(D, M') < L(D, M)$ **then**
      $\hat{M} \leftarrow M'$
      $\mathcal{Y} \leftarrow Y$
    end if
  end for
until $M = \hat{M}$
return C

B. Experiments and Data

Here, we detail the setup and training of the individual networks, and provide further experimental results. In particular, we first discuss the training setup for MNIST and highlight key results in App. Sec. B.1, and then provide additional insights into *ImageNet* prototypes. For *ImageNet*, we first shortly discuss prototypes obtained for GoogLeNet – a different network architecture than VGG – in App. Sec. B.2.1 and then proceed to show additional results on VGG-S for *ImageNet* in App. Sec. B.2.2-B.3. Finally, we show prototypes obtained for the study on fine-tuning for the Oxford Flower data that reflect the general trend observed for this data set in App. Sec. B.4.

B.1. MNIST training

We trained a CNN on MNIST 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.

In Fig. 9, we show the average activation maps as background, and neurons found in a tail of a rule containing the corresponding label in its head for filter 2 in the first convolutional layer. We observe that EXPLAINN discovers individual rules spanning multiple classes that describes pixel groups that detect common areas of a set of numbers, such as the top left stroke in 4s and 9s. The average activation maps as visualized in the same figure cannot reveal such fine-grained information, neither can do a prototype for the filter (see Fig. 10).

Another example is given in Fig. 11, where we visualize the 36th filter in the second convolutional layer. We observe that the discovered rules indicate the role of the filter to be a horizontal edge detector, with shared features, such as the top stroke of 0,2,3,5,8, and 9, being captured by the same part of the filter. Neither average activation maps, nor prototypes – both visualized in the same figure – are able to detect this behaviour, as they can only capture the global behaviour of the filter across all pixels, rather than localized pixel areas. Furthermore, without proper learning it is unclear which label combinations should be considered in unison for these two methods, whereas EXPLAINN automatically detects labels that share neuron activations by rule heads. Finally, we provide the discovered rules for filter 12 in convolutional layer 1 in Fig. 12. We observe that this filter acts as a negative, "carving" out the surroundings of the digits.

B.2. *ImageNet* further results

Here, we present additional results on the *ImageNet* data set. If not specified directly, pretrained models were obtained from the references indicated in the main manuscript.

B.2.1. GOOGLENET RESULTS

To examine if rule mining also works on different network architectures and prototyping methods, we ran GoogLeNet pretrained on *ImageNet* and gathered activation values across the network. Here, we only focus on rules from output to last hidden layer for brevity. Similar to VGG-S, we find expressive rules that span multiple classes and multiple neurons in the last layer, capturing typical structures of the classes (see Fig. 13). We observe, however, that this particular prototyping method yields harder to interpret images, which is known to be an issue and not due to the rules.

B.2.2. VGG SHARED NEURONS

One key result for the VGG-S network for *ImageNet* is that, similar to the previous MNIST network, traits that are shared between classes are encoded by the same set of neurons. We discovered many such shared traits that the network 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 the ugly face of a whole group of different dog breeds (see App. Fig. 14). So far, it is common practice to only visualize class prototypes, which can be very misleading, as shown in the next section.

B.2.3. RESOLVING THE MEANING OF CLASS PROTOTYPES

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-S network in Fig. 15, 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

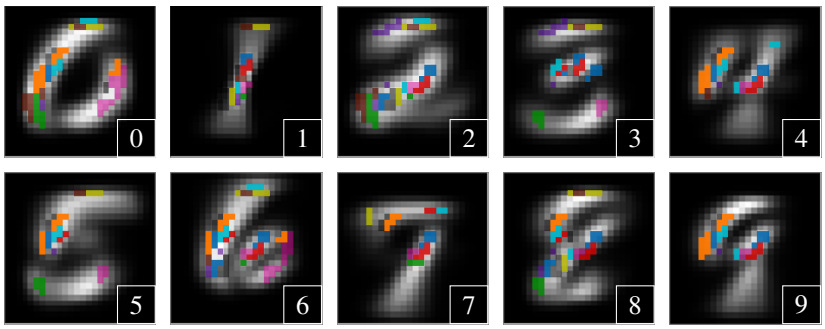


Figure 9: *MNIST* Average activation of neurons for digit classes for filter 2 in the first convolutional layer. Overlayed are the EXPLAINN rules, where pixel groups of the same colour (e.g. purple pixels top left for classes 2, 3) belong to a single rule spanning multiple classes.

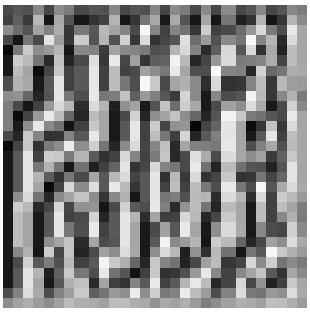


Figure 10: *MNIST prototype.* Prototype image for filter 2 from the first convolutional layer.

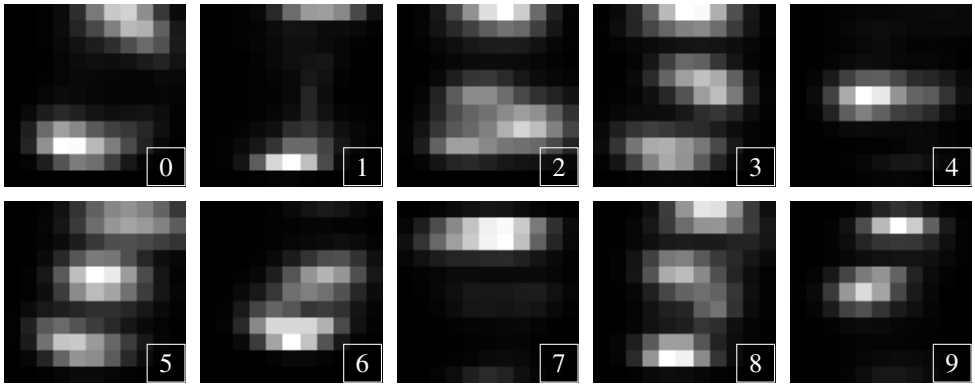
class prototype is a mixture of these different types, which explains the difficulty to interpret that prototype.

B.3. Additional results on VGG-S

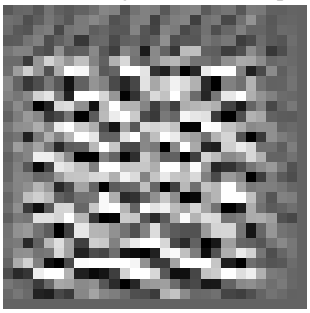
In App. Fig. 16, 17, we provide additional results based on prototyping for rules found for *ImageNet* data and the VGG-S network. We focus on rules with multiple neurons in the tail, as such class and multiclass prototypes can hardly be found by hand. Overall, we observed that the larger the number of neurons in the tail, the sharper and more interesting the resulting protoype. Furthermore, we found that for many prototypes spanning multiple classes, we discover multiple rules for some of these classes (e.g. `Black Grouse`) and the protoypes indicate that only a fraction of information, such as patterns, a colored leg or beak, or a color patch, is used from each group of neurons such that together they arrive at the class prediction.

In App. Fig. 16, the first row of the panel are examples of neuron groups that learn typical shapes of objects, such as `Sombrero` or `Gondola`. The second row contain groups of neurons capturing typical patterns and colors for individual classes, such as yellow patches on black skin of the `Fire Salamander`, red caps with white dots of the `Agaric` mushroom, the typical leaf with red veins of `Sorrel` or the wings of a `Monarch` butterfly. The third row contains common features between two classes that are together captured by the same group of neurons, like the arch-like structures and round rooftops found for certain `Triumphal Archs` and `Mosques`, the layered and intertwined worm-like shapes of many `Fur Coats` and the `Gyromira` mushroom, or the characteristic traditional covering of yurts and the front part of dogsleds.

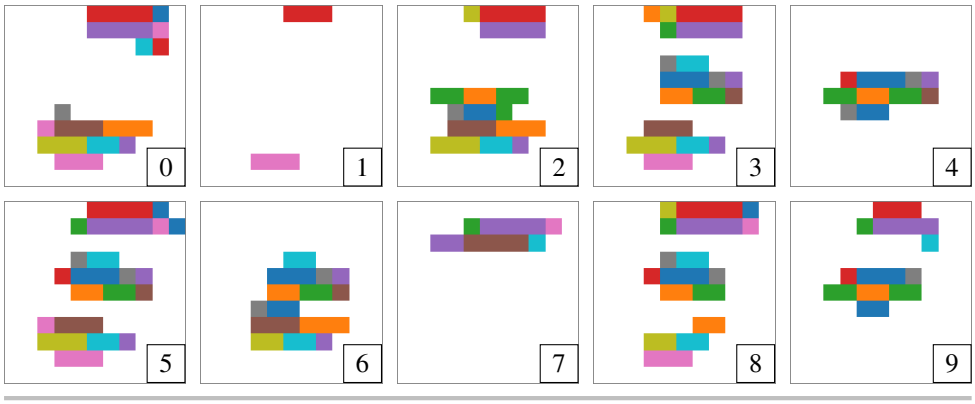
In App. Fig. 17, groups of neurons that are shared between multiple classes are visualized both revealing surprising similarities, as well as confirming that the network learns similarities that we also use as a human. In the first row, the neurons described by the first two images capture the typical shape and red color of the ears shared between the `Red Fox` and the `Lesser Panda`, respectively the insect legs and shiny turquoise color of the body of `Tiger Beetles` and `Damselflies`. Intriguingly, the network also learns a roundish shape and distinct pattern between the `Jackfruit` and the `Squirrel Monkey`. At this point, we would like to invite the reader to look up how the top of the head of such a monkey looks like, it resembles surprisingly well the size, color, shape, and texture of a Jackfruit. For the last picture in the first row of this panel, we see dotted wings that clearly are related with the associated labels `Cabbage Butterfly`, and `Sulphut Butterfly`. But opposed to visualizations related to other Butterflies (given in both panels), the wings are all oriented in a distinct way, which resemble the cap of a dotted mushroom, which might explain the association with the `Agaric` mushroom. In the second row, we observe that the network captures common features shared between similar classes - in this case closely related animals - with the same set of neurons, which matches human intuition.



(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 11: *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.

B.4. Oxford Flower data

One common approach to tackle the issue of learning networks for problems with scarce training data is fine-tuning. There, networks (pre-)trained on larger, usually more general data sets, are refined on the task-specific training data, often freezing weights in earlier layers of the network and training the last layers for a few rounds, assuming that the earlier layers detect abstract features that are similar in the specific task. For example, in the earlier MNIST experiments, we saw filters detecting horizontal edges or certain strokes. For a data set on e.g. handwritten letters, such features would be similarly useful, but have to be puzzled together in a different way, which is supposedly achieved by the later layers.

Here, we look at the vanilla VGG-S network trained on *ImageNet*, and compare it to the VGG-S network fine-tuned on Oxford Flower data (see main paper for references). The Oxford Flower data consists of 8000 images of 102 flowers. We again look at rules from output to last hidden layer and report a representative set of prototypes in Fig. 18. In-

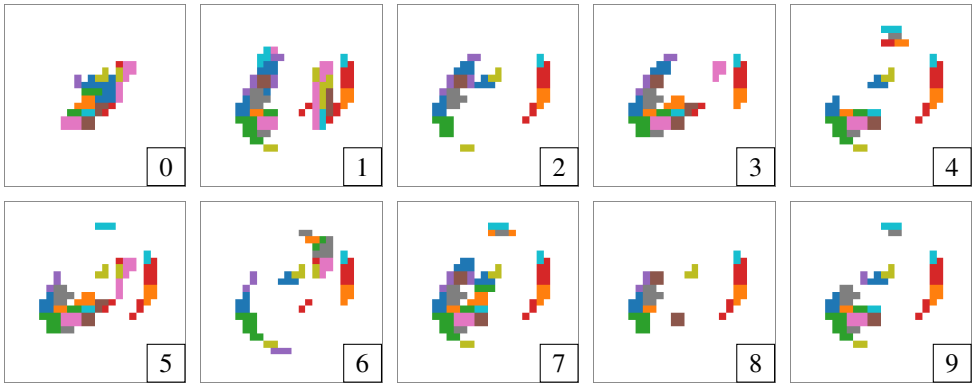
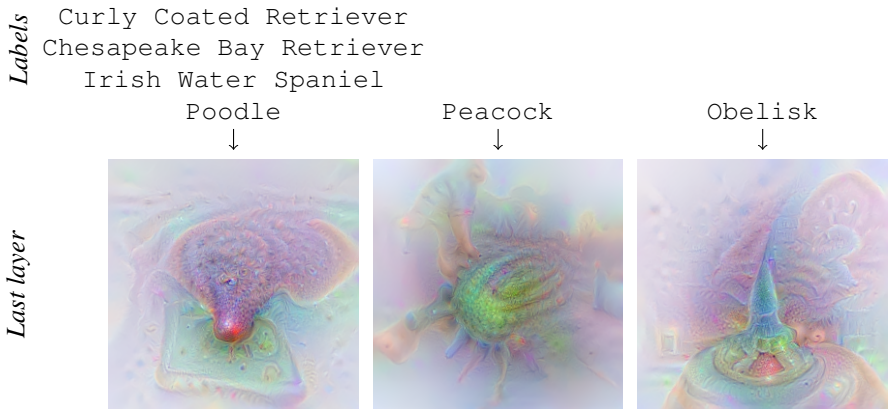


Figure 12: *The negative of a digit.* Visualizations for filter 12 in the first convolutional layer. This filter seems to capture the 'negatives' of the handwritten digits.

triguingly, we observe that when visualizing the same set of neurons of rules for the fine-tuned network also for the original network, we do find almost the same prototypes which capture the key traits of the flowers. Only minor differences can be seen with slightly more pronounced shapes and more intense colors. This is a strong indication that information about these specific flowers is already in the vanilla network hidden in some specific combination of neurons, although the network never had to classify those, nor has it probably seen these flowers in the original *ImageNet* data.



(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 13: *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).

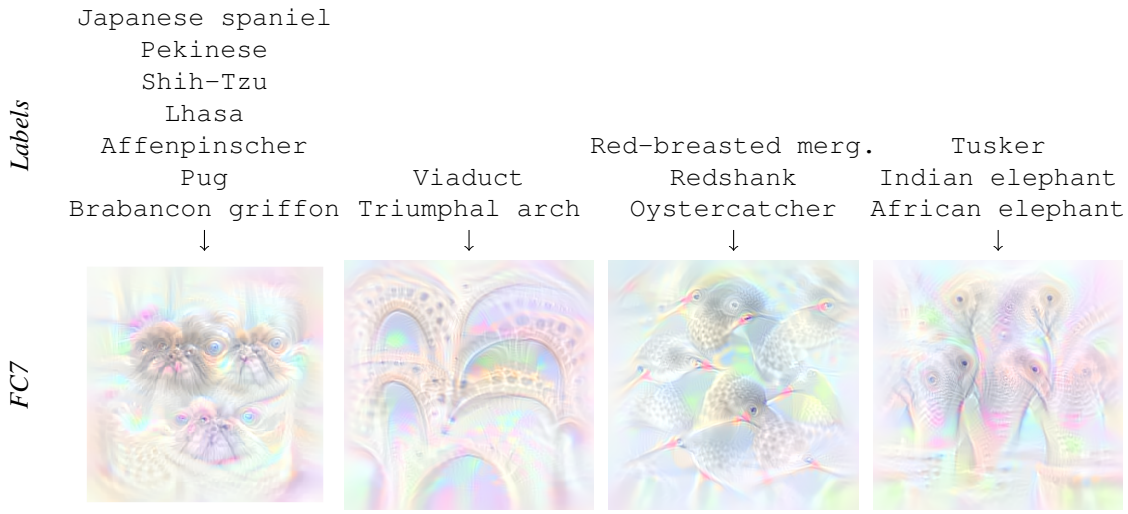


Figure 14: *Shared information across labels.* Visualizations for the rules found between the labels and the last fully connected layer (FC7). The labels in the rule heads are written above the prototype images of the tail unit groups. Each rule tail captures some interesting features of the corresponding 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.

Class prototype for `Great Danes`

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

Figure 15: 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 are 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.

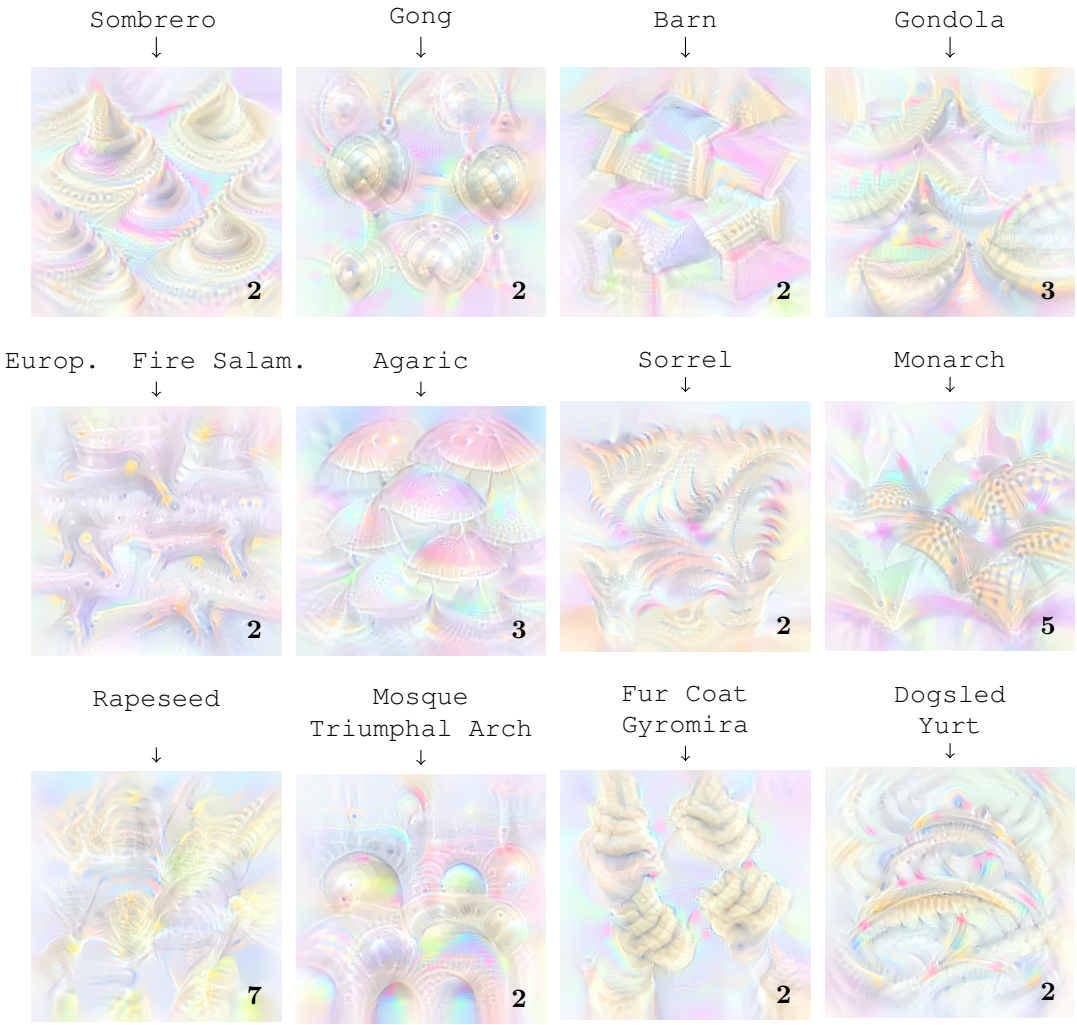


Figure 16: *Diverse prototypes*. Visualized are prototypes for rules found in the VGG-S network for *ImageNet* data between the output and last hidden layer. The class labels corresponding to the output are given above each image, the size of the group of neurons that this picture was generated from is given in the bottom right.

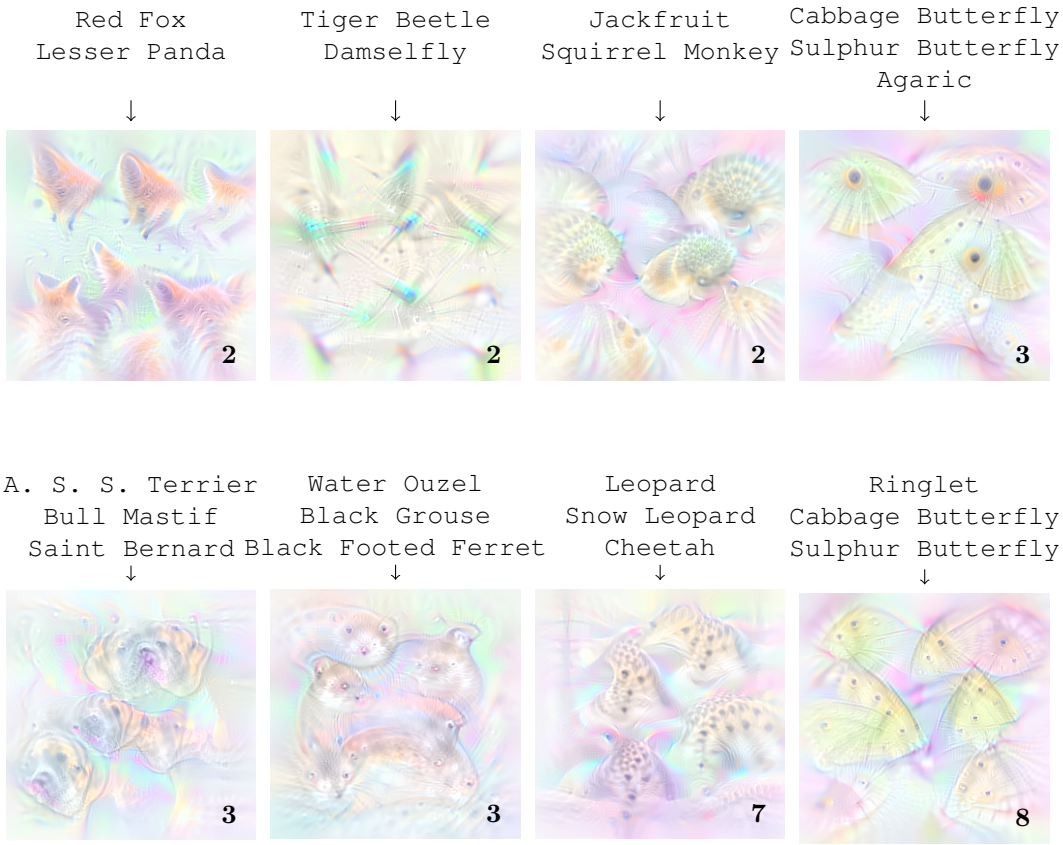


Figure 17: *More prototypes*. Visualized are prototypes for rules found in the VGG-S network for *ImageNet* data between the output and last hidden layer. The class labels corresponding to the output are given above each image, the size of the group of neurons that this picture was generated from is given in the bottom right.

Figure 18: *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).