On Modifying the Perception of a Neural Network

Manuel de Sousa Ribeiro, João Leite

NOVA LINCS, NOVA School of Science and Technology, NOVA University Lisbon, Portugal

Abstract

Artificial neural networks are typically regarded as black boxes, given how difficult it is for humans to interpret how these models reach their results. One way in which humans attempt to interpret complex systems is by imagining how they behave in hypothetical scenarios. In this work, we propose a method that allows one to modify what an artificial neural network is perceiving regarding specific human-defined concepts of interest, allowing one to test how they behave under such hypothetical scenarios. Through empirical evaluation, we test the proposed method on different models and datasets, assessing its qualities.

Keywords

Neural Networks, Interpretability, Explainability, Counterfactual

1. Introduction

In this paper, we investigate how to influence neural network models to identify specific human-defined concepts of interest not directly encoded in their inputs, with the ultimate goal of better understanding how such concepts affect a model's predictions. Our results suggest that this is possible with little labeled data and without the need to change or retrain the existing neural network model.

As neural networks start to be applied in critical domains, their lack of interpretability [1] becomes a central problem. This lack of interpretability stems from the fact that, apart from their internal activations – which lack any declarative meaning – neural networks do not provide any indication to support their results [2].

Recent methods to help interpret artificial neural network models – e.g., saliency and attribution methods [3, 4, 5, 6, 7, 8], or proxy-based methods [9, 10, 11, 12, 13, 14] – mostly do so in terms of the inputs of the model being interpreted, with explanations essentially consisting of sets of input features and their corresponding contributions.

However, various user studies [15, 16, 17] have shown that explanations such as the ones given by these methods are often disregarded or unhelpful to end users. One reason is that humans do not typically reason with features at a very low level, such as the individual pixels of an image, but rather with higher-level human-defined concepts of interest, which are not necessarily directly available in the input. E.g., humans would typically expect an explanation of why a particular picture of a train was classified as being of a 'freight train' to be based on the presence of a 'freight wagon', and the absence of a 'passenger car', rather than a list of relevant pixels.

The need to pay attention to these higher-level human-defined concepts of interest[1], together with the research conducted in the field of neuroscience in identifying highly selective neurons known as *concept cells* that seem to "provide a sparse, explicit and invariant representation of concepts" [19, 20], inspired a recent area of research generally known as concept-based explainable AI [21, 22]. This research area is based on the idea of peeking into the activations of groups of neurons to comprehend their function and what they encode, resulting in methods such as TCAV [23], Concept Whitening [24], and Mapping Networks [25].

Whereas these methods soundly contribute to the development of better explanations for neural networks – by bringing into play higher-level human-defined concepts of interest not directly available

MAI-XAI'25: Multimodal, Affective and Interactive eXplainable Artificial Intelligence, October 25–26, 2025, Bologna, Italy
mad.ribeiro@fct.unl.pt (M. de Sousa Ribeiro); jleite@fct.unl.pt (J. Leite)
0000-0002-5526-1043 (M. de Sousa Ribeiro); 0000-0001-6786-7360 (J. Leite)

[1] In the literature, these *concepts* are sometimes referred to as *properties* [18].

in the input – they may still be insufficient in providing humans the insight they usually need to properly understand a model. It is well known that humans often attempt to determine the causes for non-trivial phenomena by resorting to counterfactual reasoning [26], i.e., trying to understand how different scenarios would lead to different outcomes. E.g., a child might try to understand what a 'freight train' is by asking whether the train would still be considered a freight train if it was carrying passengers, instead of cargo. This approach seems to be helpful for interpreting neural networks [27], as it emphasizes the causes – what is changed – and the effects – what mutates as a result of the changes. For example, to better interpret how a neural network is classifying a particular image of a train, the user could ask what would have been the output had the model identified a passenger car instead of a freight wagon.

There has been work on developing methods to generate counterfactuals for artificial neural networks [28], with some allowing for generating counterfactual samples with respect to particular attributes [29]. However, such methods are typically complex to implement, often requiring the training of an additional model to produce the counterfactuals [30], or the use of specific neural network architectures, e.g., invertible neural networks [31], which is at odds with our goal of understanding existing (deployed) models, hence not modifiable. But, more importantly, these methods focus on particular changes to the input samples, neglecting what the model is actually perceiving from those inputs, which is inadequate when at the center of the counterfactual is a concept not directly encoded in the input. When questioning whether a neural network model would still consider a train to be a 'freight train' had it been carrying passengers, it is not sufficient to modify the image to include a passenger car, since the model may still misinterpret the modified image. What we are interested in knowing is whether *perceiving* a passenger car would have changed the model's output.

In this work, we address the issue of counterfactual generation at a different level of abstraction, focusing on what a model perceives from a given input regarding specific human-defined concepts of interest and how that affects the model's output. Given the evidence that neurons similar to *concept cells* seem to emerge in artificial neural networks [32], and inspired by the technique of optogenetics which allows the manipulation of the activity of specific neurons to learn their purpose [33], we propose and test a method to generate counterfactuals by manipulating the outputs of the neurons of an artificial neural network associated with the concepts of interest, without producing a specific corresponding input. Our method is based on the idea of "*convincing*" the model that it is perceiving a particular concept of interest, for example, a passenger wagon, rather than generating a modified image containing one.

By abstracting from the specific input sample and instead focusing on generating counterfactuals by modifying what a model perceives with respect to a particular concept, we avoid commiting to a concrete instantiation of that concept in the input sample. In this way, we can focus directly on what a model is perceiving and how changing it impacts the model's predictions in a human-understandable manner.

Our results show that by changing the activations of these concept-like neurons using our method, the model *reacts* as if it had indeed perceived that concept, changing its output accordingly.

In Section 2, we present the proposed method, with Section 2.1 discussing how to pinpoint which neurons identify a particular concept of interest in a model, and Sections 2.2 to 2.4 addressing different aspects of the method and providing experimental evidence to support our claims. Section 3 illustrates two alternative applications of the method. In Section 4, we apply and test the method in two real-world datasets. In Section 5, we discuss related work, concluding in Section 6.

2. Method

In order to generate counterfactuals regarding what a neural network model is perceiving about human-defined concepts of interest, we propose a method composed of three main steps. Namely, for a given concept of interest:

a) Estimate how sensitive each neuron is to that concept, i.e., how well its activations separate

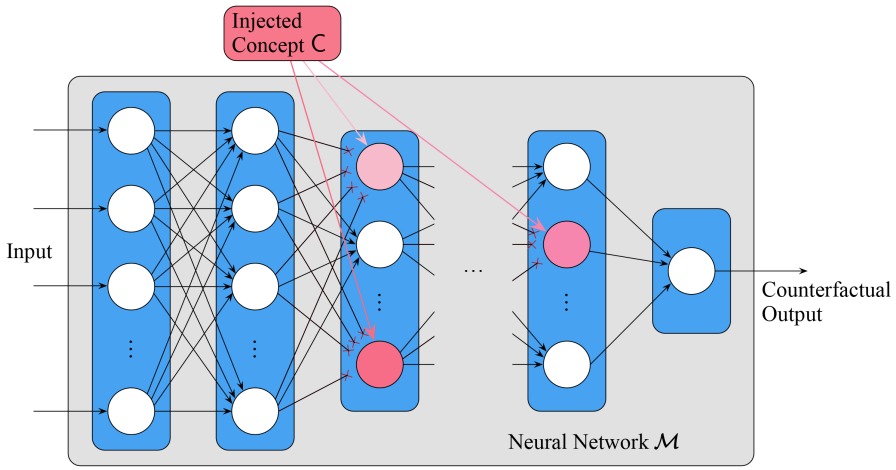

Figure 1: Injecting some concept C in a neural network model.

samples where that concept is present from those where it is absent;

b) Based on the neurons' sensitivity values, select which neurons are considered as "concept cell-like", which we will refer to as *concept neurons*;

c) For each concept neuron, compute two activation values, representing, respectively, the output of that neuron for samples where that concept is present and absent.

Then, to generate a counterfactual output with regard to the considered concept of interest, one only has to modify the activations of the selected units to the respective newly computed ones.

Consider a neural network model $\mathcal{M} : \mathbb{I} \to \mathbb{O}$, which is the model being analyzed, where $\mathbb{I}$ is the input data space and $\mathbb{O}$ denotes the model output space. For each human-defined concept C, we assume a set of positive $P_{\mathsf{C}} \subseteq \mathbb{I}$ and negative $N_{\mathsf{C}} \subseteq \mathbb{I}$ samples where the concept is, respectively, present and absent. These sets provide an extensional definition of concept C, conveying its meaning. Let $U_{\mathcal{M}}$ be the set of neurons of model $\mathcal{M}$. Then, for each neuron $i \in U_{\mathcal{M}}$, we consider $a_i^{\mathcal{M}} : \mathbb{I} \to \mathbb{R}$ to be a function providing the output of neuron i of $\mathcal{M}$.

The first step of our method consists of estimating how sensitive each neuron is to some concept C, i.e., how well the activations of each neuron separate samples where a concept is present from those where it is not. We denote by $r_i^{\mathcal{M}} : 2^{\mathbb{I}} \times 2^{\mathbb{I}} \to [0, 1]$ the function which, for a given neuron $i \in U_{\mathcal{M}}$, takes a set P_{C} and N_{C} and provides a value representing how sensitive i is to concept C. In Section 2.1, we consider different implementations of this function.

The second step is to select which neurons of $\mathcal{M}$ are to be considered as concept neurons for C. This set is defined as $S_{\mathsf{C}} = \{i : r_i^{\mathcal{M}}(P_{\mathsf{C}}, N_{\mathsf{C}}) > \alpha_{\mathsf{C}}\}$, where α_{C} is a sensitivity threshold associated with concept C. One might vary this threshold to observe how the model reacts when different sets of neurons are selected as concept neurons. In the remaining of the paper, we finetune each sensitivity threshold by using 20% of the samples in P_{C} and N_{C} as a validation set, iteratively decreasing α_{C} until 15 neurons are added with no improvement. To estimate the performance of a sensitivity threshold, the samples in the validation set should be classified with respect to the expected output of $\mathcal{M}$ given the presence or absence of C.

Lastly, for each selected neuron in S_{C}, we compute two activation values representing, respectively, when that concept is present and absent. This is captured by a function $c_i^{\mathcal{M}} : 2^{\mathbb{I}} \to \mathbb{R}$ that computes an activation value for neuron $i \in U_{\mathcal{M}}$, which is then used to compute both an activation value representing the presence of concept C, $c_i^{\mathcal{M}}(P_{\mathsf{C}})$, and one representing its absence, $c_i^{\mathcal{M}}(N_{\mathsf{C}})$. In Section 2.2, we discuss different implementations of $c_i^{\mathcal{M}}$ and compare them.

Subsequently, when some input $x \in \mathbb{I}$ is fed to neural network $\mathcal{M}$, to generate a counterfactual scenario where concept C is present or absent, we only need to replace the activation value $a_i^{\mathcal{M}}(x)$ of

Figure 2: Sample images of the XTRAINS dataset.

$$\begin{aligned}
&\mathsf{TypeA} \equiv \mathsf{WarTrain} \sqcup \mathsf{EmptyTrain} && \exists \mathsf{has.FreightWagon} \sqcap \exists \mathsf{has.PassengerCar} \sqcap \exists \mathsf{has.EmptyWagon} \sqsubseteq \mathsf{MixedTrain}\\
&\mathsf{TypeB} \equiv \mathsf{PassengerTrain} \sqcup \mathsf{LongFreightTrain} && \exists \mathsf{has.}(\mathsf{PassengerCar} \sqcap \mathsf{LongWagon}) \sqcup (\geq 2\ \mathsf{has.PassengerCar}) \sqsubseteq \mathsf{PassengerTrain}\\
&\mathsf{TypeC} \equiv \mathsf{RuralTrain} \sqcup \mathsf{MixedTrain} && \exists \mathsf{has.ReinforcedCar} \sqcap \exists \mathsf{has.PassengerCar} \sqsubseteq \mathsf{WarTrain}\\
&\mathsf{LongFreightTrain} \equiv \mathsf{LongTrain} \sqcap \mathsf{FreightTrain} && (\geq 2\ \mathsf{has.LongWagon}) \sqcup (\geq 3\ \mathsf{has.Wagon}) \sqsubseteq \mathsf{LongTrain}\\
&\mathsf{EmptyTrain} \equiv \forall \mathsf{has.}(\mathsf{EmptyWagon} \sqcup \mathsf{Locomotive}) \sqcap \exists \mathsf{has.EmptyWagon} && (\geq 2\ \mathsf{has.FreightWagon}) \sqsubseteq \mathsf{FreightTrain}
\end{aligned}$$

Figure 3: Subset of the XTRAINS dataset ontology's axioms, describing how the trains' representations are classified.

each neuron i in S_{C} with $c_i^{\mathcal{M}}(P_{\mathsf{C}})$ or $c_i^{\mathcal{M}}(N_{\mathsf{C}})$, respectively. We refer to this step, as *injecting* a concept into a neural network model, and illustrate it in Figure 1. The result of the proposed method might be observed in the model's output, which now reflects a counterfactual scenario – the model's response to some input $x \in \mathbb{I}$ conditioned by the performed injection.

Performing a thorough and systematic testing of this method requires a dataset with additional labels regarding the human-defined concepts of interest related to the task of a neural network model. Furthermore, the dataset needs to provide a description of how these labels are related to the labels of the model's output, allowing for an understanding of what would be expected of the model given the injection of some concept. Unfortunately, there are no popular benchmarks meeting such criteria, nor many datasets with these features. One notable exception is the Explainable Abstract Trains Dataset (XTRAINS) [34]. This is a synthetic dataset composed of varied representations of trains, such as those shown in Figure 2, inspired by the work of J. Larson and R. S. Michalski in [35]. This dataset contains labels regarding various visual concepts and a logic-based ontology describing how these concepts are related. Figure 3 shows a subset of the dataset's ontology illustrating how different concepts relate to each other. This ontology specifies, for example, that TypeA is either WarTrain or EmptyTrain, and that WarTrain encompasses those having a ReinforcedCar and a PassengerCar. These concepts have a visual representation, e.g, a ReinforcedCar is shown as a car having two lines on each wall, such as the first two cars of the leftmost sample in Figure 2. In the remainder of this section, we discuss and thoroughly validate the method in the setting of the XTRAINS dataset. The method is further tested using real-world data, although with a more limited set of concepts, in Section 4.

In the experiments described below, we adopt a neural network developed in [36], trained to identify trains of TypeA – referred to as $\mathcal{M}_{\mathsf{A}}$, which achieves an accuracy of about 99% in a balanced test set of $10\,000$ images. All experiments use $|P_{\mathsf{C}}| = |N_{\mathsf{C}}| = 200$ samples and were run 5 times reporting the average result on 400 test samples, unless stated otherwise. We consider the definition of relevancy given in [25] to establish which concepts are related to the task of a given neural network (w.r.t. the dataset's accompanying ontology).

2.1. Identifying *concept cell-like* neurons

To modify a neural network model's perception regarding some human-defined concept of interest, it is first necessary to determine which neurons in the model are identifying that concept. Based on the evidence that neural networks seem to have information encoded in their internals regarding concepts that are related to their tasks [25], and that neurons with "concept cell-like" properties seem to emerge in neural networks [32], we hypothesize that neural networks have neurons which act as concept cells for concepts related with the tasks they perform. In this section, we investigate how to determine such neurons in a neural network model.

We consider a neuron to be a concept neuron for some concept C based on how it separates samples

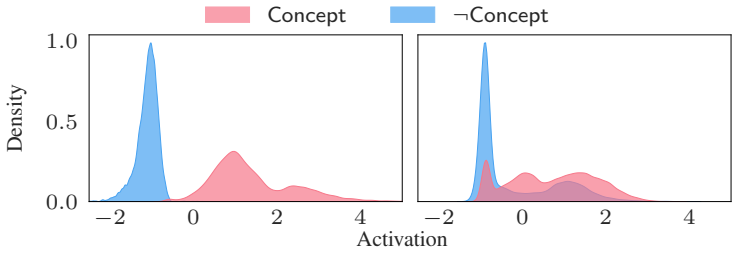

Figure 4: Probability density functions of two neurons for samples where a given concept is present and absent.

where C is present from those where it is absent. Figure 4, illustrates the estimated probability density function of two neurons for some P_{C} and N_{C}. The neuron on the left behaves as a concept neuron for C, showing well-separated distributions for both sample sets. On the other hand, the neuron on the right does not behave as a concept neuron for this concept, given that its activations do not clearly distinguish both sets.

While methods such as TCAV [23] and Mapping Networks [25] allow for an understanding of whether a given model is sensitive to a certain concept, here we are interested in understanding whether individual neurons are sensitive to that concept. Our goal is not to check whether the model is sensitive to a concept, for which only some neurons might be sufficient, but rather to find all neurons that are sensitive to that concept and manipulate their activations.

We consider three implementations of $r_i^{\mathcal{M}}$ to evaluate the adequacy of a neuron i of $\mathcal{M}$ as a concept neuron for C which provide different approximations for how well separated two distributions are while having different assumptions regarding the underlying data distributions and different computational cost. They are:

- Spearman rank-order correlation between i's activations and C's presence, computed as $|1 - \frac{6\sum d_j^2}{n(n^2-1)}|$, where $d_j = a_i^{\mathcal{M}}(P_{\mathsf{C}j}) - a_i^{\mathcal{M}}(N_{\mathsf{C}j})$ and $n = |P_{\mathsf{C}}| = |N_{\mathsf{C}}|$;
- Accuracy of a linear binary classifier (b) predicting C's presence from i's activations, computed as $\frac{|\{x : x \in N_{\mathsf{C}} \cap b(a_i^{\mathcal{M}}(x)) = 0)\}| + |\{x : x \in P_{\mathsf{C}} \cap b(a_i^{\mathcal{M}}(x)) = 1)\}|}{|N_{\mathsf{C}}| + |P_{\mathsf{C}}|}$;
- Area of intersection between the probability density function curves for i's activations in P_{C} and N_{C}, computed as $1 - \int min(f_i^{P_{\mathsf{C}}}(x), f_i^{N_{\mathsf{C}}}(x))\, dx$, where $f_i^{P_{\mathsf{C}}}$ and $f_i^{N_{\mathsf{C}}}$ are the estimated probability density functions for i's activation values in P_{C} and N_{C}, respectively.[2]

To test our hypothesis that "concept cell-like" neurons exist in neural network models for concepts that are related to the task a model is performing, we compute the three metrics described above for four random concepts that are relevant to the task performed by $\mathcal{M}_{\mathsf{A}}$: EmptyTrain, ∃has.PassengerCar, ∃has.ReinforcedCar, and WarTrain; and for four random non-relevant concepts: ∃has.LongWagon, ∃has.OpenRoofCar, LongFreightTrain, and MixedTrain.

According to our hypothesis, we would expect to find neurons for the relevant concepts with high values in the described metrics, while for non-relevant concepts we should be unable to do so.

Table 1 shows the mean and standard deviation of the top 20 most sensible neurons identified by that metric for each concept and each implementation of $r_i^{\mathcal{M}}$. For instance, the 20 neurons most sensible to the EmptyTrain concept allow on average for this concept to be predicted from their activations with an accuracy of 99% using a linear binary classifier. It is observable that while sensitive neurons can be identified for all of the relevant concepts, as indicated by the high sensitivity values, for the non-relevant concepts we are unable to find such neurons, as witnessed by the low sensitivity values.

While the number of neurons that are sensible to each concept will vary depending on the specific sensitivity threshold α_{C} used, for the concepts we considered it varied between 10 and 40. Considering that $\mathcal{M}_{\mathsf{A}}$ contains about 2×10^6 neurons, the amount of selected concept neurons is minuscule in

[2]Probability density functions are estimated using kernel density estimation, and integrals using Simpson's rule.

Table 1
Concept sensibility (mean and standard deviation) for relevant and non-relevant concepts.

	Concept	Spearman	Accuracy	Intersection
Relevant	(a) EmptyTrain	0.85 ± 0.02	0.99 ± 0.01	0.95 ± 0.04
	(b) ∃has.PassengerCar	0.86 ± 0.01	0.98 ± 0.01	0.95 ± 0.03
	(c) ∃has.ReinforcedCar	0.86 ± 0.01	0.99 ± 0.01	0.96 ± 0.02
	(d) WarTrain	0.85 ± 0.02	0.97 ± 0.02	0.87 ± 0.07
Non-Rel	(e) ∃has.LongWagon	0.19 ± 0.01	0.6 ± 0.01	0.22 ± 0.01
	(f) ∃has.OpenRoofCar	0.20 ± 0.02	0.61 ± 0.01	0.23 ± 0.01
	(g) LongFreightTrain	0.35 ± 0.03	0.67 ± 0.01	0.32 ± 0.01
	(h) MixedTrain	0.59 ± 0.03	0.84 ± 0.01	0.67 ± 0.03

comparison. Interestingly, most selected neurons belong to the dense part of the model, with those corresponding to more abstract and complex concepts appearing in the later dense layers. Section 2.2 explores the importance of the selected concept neurons for the overall method's performance.

These results seem to support our hypothesis, indicating that some of the neurons in a neural network model behave as concept cells for specific concepts related with the task of that neural network. We note that while this does not imply that it is always possible to find such neurons for any given concept, it shows that neurons with such behavior do arise in neural network models.

2.2. Manipulating a neural network's perception

Being able to identify concept neurons in a model, we now test our main hypothesis: – that it is possible to manipulate a neural network's perception regarding specific human-defined concepts of interest by manipulating the activations of their respective concept neurons.

If our hypothesis holds, we expect that when injecting a given concept into a model, the output of the model reflects the performed injection. For instance, if we feed an image of a train having a passenger car and no reinforced cars to $\mathcal{M}_\mathsf{A}$, which was trained to identify TypeA trains, we would expect it to output that this is not a TypeA train (cf. ontology in Figure 3). However, if we identify the concept neurons for ∃has.ReinforcedCar – having a reinforced car – and modify their outputs to indicate the presence of a reinforced car, i.e., we *inject* the concept ∃has.ReinforcedCar, we expect the model to change its output and identify a TypeA train. To analyze the performance of the method in the setting of the XTRAINS dataset, we consider, for each of the relevant concepts C analyzed in Section 2.1, four different test sample sets according to the following criteria:

- S_1 - injecting C should not change $\mathcal{M}_\mathsf{A}$'s output.
- S_2 - injecting C should change $\mathcal{M}_\mathsf{A}$'s output.
- S_3 - injecting ¬C should not change $\mathcal{M}_\mathsf{A}$'s output.
- S_4 - injecting ¬C should change $\mathcal{M}_\mathsf{A}$'s output.

To determine whether the output of $\mathcal{M}_\mathsf{A}$ should change, we reason with the ontology provided with the dataset.

We consider three alternatives to compute the new activation values for each concept neuron ($c_i^\mathcal{M}$): computing the median of the neuron's activations; computing the mode; and using the method from [37], which suggests the use of an auxiliary probe model trained to predict whether a concept is present in the activations of a trained neural network and computes new activation values that lead to the desired output of the probe.

Table 2 shows the average accuracy of the method for the four considered test sets using the different implementations of $r_i^\mathcal{M}$ and $c_i^\mathcal{M}$. The high percentage of samples where the model's output changed as expected, given the performed injection, provides evidence that it is possible to manipulate a neural

Table 2
Neurons and correctly classified samples by $r_i^{\mathcal{M}}$ and $c_i^{\mathcal{M}}$.

	Spearman		Accuracy		Intersection	
	#	%	#	%	#	%
Median	24	92.2	26	98.0	16	98.0
Mode	21	87.3	21	95.5	18	97.3
[37]	17	88.6	15	91.6	11	89.7

Table 3
Correctly classified samples in each test set.

Concept	Neurons	S_1 (%)	S_2 (%)	S_3 (%)	S_4 (%)
(a) EmptyTrain	14	100.0	99.6	98.4	100.0
(b) ∃has.PassengerCar	21	98.8	92.8	99.0	91.0
(c) ∃has.ReinforcedCar	18	99.0	99.8	99.4	92.0
(d) WarTrain	11	100.0	97.6	99.8	100.0

networks' perception regarding different human-defined concepts by modifying the activations of specific neurons which seem to identify those concepts. Furthermore, these results suggest that the sensitivity metric based on the area of the intersection between the probability density functions of a neuron's activations for P_{C} and N_{C}, when using the median value of the neurons' activations, provides better results, while modifying the activations of fewer neurons. For this reason, throughout all remaining experiments, we adopt this metric to compute $r_i^{\mathcal{M}}$, and adopt $c_i^{\mathcal{M}}$ as the median value.

Table 3 shows the percentage of samples where injecting a concept resulted in the expected change in $\mathcal{M}_{\mathsf{A}}$'s output for each of the sets described above. Overall the results seem to be quite positive, indicating that typically the model outputs the expected result given the concept being injected. The result of injecting ∃has.PassengerCar in set S_2 and its negation in set S_4 seem to be somewhat inferior. While inspecting these samples, we found that this seems to be due to $\mathcal{M}_{\mathsf{A}}$ not having properly captured the relationship between the concept of ∃has.PassengerCar and the concept of EmptyTrain. We further explore this matter in Section 3.

2.3. Importance of the selected neurons

In this section, we investigate how dependent the proposed method is on the sensitivity threshold α_{C}. We vary the sensitivity threshold α_{C} for each concept C, while examining the number of concept neurons it selects and how they affect the method's performance.

Figure 5 shows the distribution of the 10 best performing sets of concept neurons obtained when varying the sensitivity threshold α_{C} for each of the four relevant concepts analyzed in Section 2.1, with respect to the four test sets introduced in Section 2.2. For instance, Figure 5 shows that the injection of concept EmptyTrain performs best between 13 to 26 neurons, with its median at about 20 neurons.

Figure 5 also shows that each concept has a different optimal range for its sensitivity threshold α_{C}, with some being broader than others. This highlights the importance of finetuning the sensitivity threshold α_{C} for each concept C. Finetuning the sensitivity threshold α_{C} might be done by taking into account the number of neurons that are selected as concept neurons in set S_{C} and gradually considering additional neurons until the method's performance stops increasing. This can be done using a small validation set – here we consider 80 validation samples. Section 2.4 illustrates the usage of fewer samples.

Also, by inspecting the results for each test set, we observed that when the sensitivity threshold α_{C} is too high – excluding neurons that act as concept cells – the method is unable to produce the desired change in the model's predictions, resulting in the output of the model being unaltered (poor performance in the test sets S_2 and S_4). The performance also degrades for low sensitivity thresholds

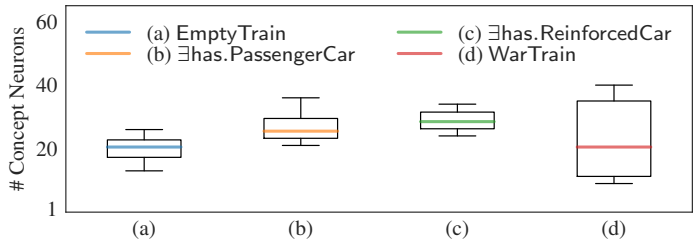

Figure 5: Distribution of best-performing sets of concept neurons.

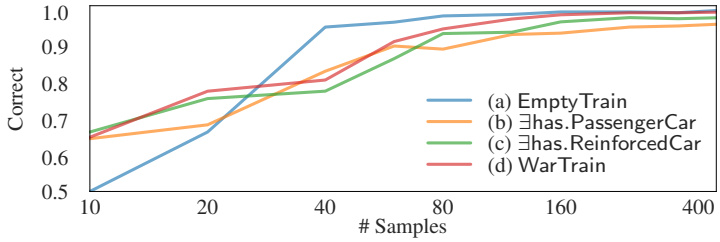

Figure 6: Correct samples by amount of available data.

since neurons that do not behave as concept cells become selected.

2.4. Counterfactual's cost

The proposed method has two main costs: the data in P_{C} and N_{C}; and the labeling of the samples used to finetune α_{C}. In this section, we verify the feasibility of the proposed method regarding its costs, i.e., the amount of data required to modify a model's perception regarding some concept C.

Figure 6, shows the average percentage of samples – from the four test sets introduced in Section 2.2 – where the output of the model is consistent with the performed manipulation for different quantities of available data. The number of samples shown corresponds to the total amount of used samples (including validation data).

We can observe that even with as few as 60 samples, the method is able to manipulate $\mathcal{M}_{\mathsf{A}}$'s perception with success in more than 90% of the test set samples for the considered concepts. The performance seems to stop improving after around 300 samples.

Whereas, as expected, the method's performance seems to drop significantly when the amount of data is insufficient to properly identify the concept neurons, the low cost associated with the fairly low amount of data required for the method to properly function, as shown by the experiments, seems to suggest that it may be applicable in settings with limited available data.

3. Applications and other uses

The previous section described a method to generate counterfactuals for a neural network model by modifying the activations of neurons which act as concept cells for human-defined concepts of interest. This method allows for an understanding of how different concepts influence the output of a model under different scenarios. What if one is interested in understanding how a model relates different concepts of interest that are not represented in the model's output? Or what if one knows a model is providing an incorrect result and wants to provide additional information? In this section, we illustrate the application of the proposed method to address these two questions.

Interpreting neural networks

Consider the situation introduced in Section 2.2, where we hypothesize that $\mathcal{M}_\mathsf{A}$ may not have properly captured the relationship between the concepts of $\exists$has.PassengerCar and EmptyTrain. Although both of these concepts are not represented in $\mathcal{M}_\mathsf{A}$'s output, they are both relevant for its task, and one might be interested in ensuring that the model understands the relationship between both.

To understand whether a given concept that is not represented in the output of a model is being identified by it, we use the *mapping networks* from [25]. These are small probe neural networks trained to discern whether a given human-defined concept of interest is present in the activations of a neural network model. Mapping networks receive as input the activations of a trained neural network model, and output whether a specific human-defined concept of interest is present in those activations. Thus, when an input sample is fed to a neural network model, we can use a mapping network to verify whether a specific concept of interest is identified by that model.

To verify whether a model has captured the relationship between two concepts, one might inject one of the concepts and use a mapping network to assess how the other concept reacts to the performed injection. For instance, if $\mathcal{M}_\mathsf{A}$ captured that empty trains do not have passenger cars, when the concept of EmptyTrain is injected, a mapping network for $\exists$has.PassengerCar should indicate the absence of passenger cars. Similarly, if $\mathcal{M}_\mathsf{A}$ has captured that if a train has a passenger car, then it is not an empty train, when $\exists$has.PassengerCar is injected, a mapping network for EmptyTrain should indicate the absence of an empty train.

To test this, we train mapping networks for EmptyTrain and $\exists$has.PassengerCar – both achieving an accuracy of about 99% on a 1000 balanced test set. By injecting concept EmptyTrain on 1000 samples of trains with passenger cars, we verify that in 95.6% of the samples, after injection, concept $\exists$has.PassengerCar goes from being present to absent. However, when injecting concept $\exists$has.PassengerCar on 1000 samples of empty trains, in only 2% of them the concept EmptyTrain turns absent. This suggests that while $\mathcal{M}_\mathsf{A}$ has captured that empty trains do not have passenger cars, it has not picked up that if a train has a passenger car, it is not empty.

This example illustrates a possible application of the proposed method, namely to investigate whether certain relations between user-defined concepts of interest are encoded in a model. More importantly, the results suggest that the model was able to naturally integrate the injected information, impacting related concepts, as if the injected concept was truly being perceived by the model.

Correcting misunderstandings

When a neural network model provides an incorrect result, it is often difficult to interpret the cause of the error. A possible reason might be the model being unable to correctly identify what is represented in its input, in which case the proposed method might be used to "correct" the model's perception.

To test this hypothesis, we consider the use of mapping networks to identify the cause for a model's misclassification and use our proposed method to inject a concept, generating a counterfactual scenario where the misidentification is corrected. To this end, mapping networks are trained for the concepts $\exists$has.ReinforcedCar and $\exists$has.PassengerCar – with an accuracy of about 99% on a 1000 balanced test set – and used to identify whether $\mathcal{M}_\mathsf{A}$ provides wrong results due to either of these concepts being misperceived.

We consider all samples where $\mathcal{M}_\mathsf{A}$ provides a false negative, indicating that a sample input was not of TypeA when it was. We observe that in those where the concept of $\exists$has.ReinforcedCar was not identified, but present in the input, injecting $\exists$has.ReinforcedCar led to the output being corrected in 96.1% of the samples. Similarly, when considering the samples where $\exists$has.PassengerCar was not identified, but present in the input, injecting it led to the output being corrected in 98.7% of the samples.

These results provide evidence that the method is applicable even when a model yields incorrect results, allowing one to test whether different concepts might be responsible for the incorrect result. This enables users to further understand why their models achieved an incorrect output and helps identify potential flaws in a model.

Figure 7: Image samples of different classes of traffic signs.

Table 4
Correctly classified samples in each test set.

Concept	Neurons	S_1 (%)	S_2 (%)	S_3 (%)	S_4 (%)
Black Symbol	148	97.5	91.4	99.3	96.0
Black Band	43	100.0	98.3	99.9	94.7

4. Validation with real-world data

So far, we have tested our method in the setting of a synthetic dataset. We now validate whether we can replicate similar results in the setting of two real-world datasets, the German Traffic Sign Recognition Benchmark Dataset (GTSRB) [38] and ImageNet [39].

German Traffic Sign Recognition Benchmark dataset

Using the GTSRB dataset, we train a VGG model to distinguish four different types of traffic signals: road closed; other prohibitory signs; end of all restrictions; and the end of specific restrictions. These classes are, respectively, illustrated in Figure 7. The trained model achieves an accuracy of 99% in the GTSRB test set.

To test the method in this setting, we consider the concepts of 'having a black symbol' and 'having a black band' described in the Vienna Convention on Road Signs and Signals [40]. For each concept, we define four test sets in the same manner as those described in Section 2.2, allowing for an understanding of whether the output of the model is changing according to the performed injection. For instance, one could expect that if a black symbol is added to a road closed sign, the model would modify its output to predict some other prohibitory sign; or that if a black bar is added to a road closed sign, the model would modify its output to predict the end of all restrictions sign. We consider $|P_{\mathsf{C}}| = |N_{\mathsf{C}}| = 200$ samples and test our method with the GTSRB test set.

Table 4 shows the results for each concept and test set. The high percentage of samples where the model's output changed as expected given the performed injection provides evidence that the method is applicable in a real-world data setting. We attribute the higher value of selected concept neurons for 'having a black symbol' to the variety of different symbols a prohibitory sign may have, which might have led the model to allocate more neurons to identify them.

ImageNet dataset

Using the pre-trained MobileNetV2 model from [41], and considering the setting of the ImageNet dataset [39], we both test the proposed method and illustrate its use to correct false negatives. To this end, we select a random output class: 'Rhodesian ridgeback', which is a specific dog's breed. We define the concept of 'Rhodesian ridgeback face' by selecting a set of 98 images from ImagenetNet where a Rhodesian ridgeback dog is observable and its face is centered and completely visible, as illustrated in the samples shown in Figure 8.

To test whether the concept of 'Rhodesian ridgeback face' is important for the model to be able to recognize that breed, we manually censored every 'Rhodesian ridgeback face' in the ImageNet's validation set, as shown in Figure 8, and compared the model's accuracy before and after censoring the

Figure 8: Images of 'Rhodesian ridgeback face' (left) and censored images (right).

Table 5
Model accuracy of 'Rhodesian ridgeback' class.

	Not Censored (%)	Censored (%)	Censored + Injection (%)
Top-1	70.0	36.0	64.0
Top-5	100.0	64.0	88.0

dogs' faces. The results, shown in Table 5, indicate that censoring the dogs' faces leads to a significant drop in accuracy, which seems to be evidence of its importance for the model. However, notice that even after the censoring the model is still able to correctly classify 36% of the images, indicating that although important, there are other concepts which are relevant for identifying this class. After censoring the images, we inject the concept of 'Rhodesian ridgeback face' into 109 concept neurons and observe a big increase in accuracy. This indicates that the injected concept was successful in providing some of the censored information and helping the model provide the correct classification.

To illustrate the method's ability to correct false negatives, we select all 329 samples where the model outputs a false negative for the Rhodesian ridgeback class, considering its top-1 result. The injection of the concept of 'Rhodesian ridgeback face' led 48% of these samples to output the class of Rhodesian ridgeback. If we consider the 38 samples where the model outputs a false negative considering its top-5 outputs, we were able to modify the model's output in 71.1% to the Rhodesian ridgeback class.

These results show us that even in a setting with real data, a moderately sized model ($\approx 7 \times 10^6$ neurons), and few labeled data, it is possible to identify which neurons in a model act as concept cells for a given concept and, more importantly, manipulate the neural network's perception of that concept by manipulating the activations of those neurons.

5. Related work

The last few years have seen an increase in the development of methods for interpreting artificial neural networks, with proxy-based methods being one of the most popular approaches (cf. [42]). These methods aim to substitute the model being explained for one that is inherently interpretable and which exhibits similar behavior. In contrast, our approach to generate counterfactual scenarios does not require changing or substituting the original model, which might not always be feasible or desirable. Additionally, proxy-based methods, such as LIME [11] or most automatic rule extraction algorithms [12], attempt to explain a model in terms of its input features, which is not always adequate since they might not be meaningful to some end-users or might even be subsymbolic, e.g., pixels of an image. Our method avoids this issue by relying on human-defined concepts of interest, allowing end-users to experiment with meaningful symbolic concepts that might be tailored to their needs. Automatic rule extraction algorithms, such as the one introduced in [43], allow for inducing a theory using additional concept labels. However, being only data annotations, there is no guarantee that the resulting rules will

be faithful to the neural network's internal classification process.

Another popular approach is that of saliency and attribution methods (cf. [44]), where a model's behavior is explained by attributing a value to each input feature representing its contribution to a given prediction. Although these methods might provide some insights into which features contributed most to a prediction, they do not provide clarification regarding why those contributions justify the output, leaving the burden of understanding how those contributions are related to the specific output to the user.

Some counterfactual-based methods (cf. [28]) suffer from similar issues, providing a counterfactual sample but lacking clarification of why that sample leads to a different result. We believe that counterfactual methods would benefit from abstracting away from only generating specific counterfactual samples, to describing in a human-understandable way why those counterfactuals lead to some other particular output based on how they affect the model. One approach that seems conceptually related to ours is that of [45], which produces counterfactuals by modifying the activations of a given layer to remove some property (concept). However, it is restricted to answering questions in the form of "How will the prediction of a task differ without access to some property?". Our proposal is more general, allowing one to examine how a model's prediction differs when a concept of interest is present or absent.

Concept-based approaches [21, 22] have aimed at understanding what human-defined concepts of interest are encoded in a model [46, 47, 48], whether models are sensitive to specific human-defined concepts, like TCAV [23], and how we can leverage the representations of those concepts in a model to provide explanations for its outputs [25]. These methods allow for a better understanding of what is encoded within a neural network model through human-understandable symbolic concepts, but do not allow for an understanding of how such concepts influence a model's output. Our method opens this possibility, allowing end-users to explore counterfactual scenarios by manipulating whether a neural network perceives such concepts of interest and observing how the model reacts to these scenarios.

The aforementioned literature provides plenty of evidence that neural networks' internal representations often align with human-defined concepts, not in the sense that these models encode these abstract concepts per se [49], but that their internals present statistical regularities which distinguish samples where those concepts are present/absent.

A method that combines the automatic rule extraction and neuro-symbolic approaches is presented in [36]. This method induces logic-based theories that are faithful to the internal classification process of a neural network model based on human-defined concepts [36]. However, since the method induces the theory based on a set of observations, it mostly captures correlations between the concepts of interest and the model's output and thus is unsuitable for the purpose of examining counterfactual behavior.

Recently, a set of intervention methods known as activation patching [50], which intervene on the activations of a model to inspect how specific parts of it contribute to its behavior, has gained traction. However, the way in which the interventions are executed – e.g., substituting by zeros, noise, or activations from another sample [51] – does not necessarily convey a specific human-defined concept. In contrast, our method focuses on how the representation of a human-defined concept of interest might modify a model's behavior, rather than on investigating a specific part of a model.

Our proposed method has two main limitations, which can nevertheless be mitigated by existing related work – the need for identifying concepts of interest with which to manipulate a given neural network model, and the need for some labeled data concerning such concepts. Methods like [46] and [48], which are aimed at discovering concepts that are relevant to a model's predictions and how they are related, can help users discover promising concepts of interest with which to manipulate a neural network. Regarding the labeled data, methods like Mapping Networks [25], which allow for an understanding of when a model is identifying a given concept of interest based on its activations, might provide us the necessary labels for each concept of interest. Additionally, literature on repairing neural networks [52, 53, 54], offer methods such as KnowledgeEditor [55], which modifies the information encoded in a model to fix undesired predictions, may be used to help address any issues found when inspecting how a model reacts to different counterfactual scenarios.

6. Conclusions

In this paper, we proposed a method to generate counterfactuals regarding what a neural network model is perceiving about human-defined concepts of interest, enabling users to inquire the model, and understand how its outputs are dependent on those concepts. To this end, we explored how to identify which neurons in a model are sensitive to a given concept of interest, and how to leverage such neurons to manipulate a model into perceiving that concept. Through experimental evaluation, we showed that it is possible to manipulate a neural network's perception regarding different concepts, requiring few labeled data to do so, and without needing to change or retrain the original model.

We provided a formalization of the proposed method and illustrated how it can be applied to generate counterfactuals, but also how to use it to better understand how different concepts are related within a model, and how to inspect and correct a model's misclassifications.

The two showcased applications of the proposed method illustrate its versatility and how it opens further possibilities for new ways to inspect and interpret neural network behavior in a human-understandable manner. The first application shows how users might explore the relations between two user-defined concepts of interest encoded in a neural network model by manipulating the perception of a model regarding one and examining how the other reacts to the performed manipulation. The second application demonstrates how the method might be used for correcting a model's predictions, allowing users to straightforwardly override some aspect of the model's internal predictive process.

We conclude that for some concepts that are relevant to a neural network's task, it is often the case that there exist specific neurons that encode such concepts and are responsible for their identification within the model. Modifying the activations of these neurons, similarly to stimulating concept cells in the human brain, allows one to *trick* the model into perceiving that concept. This simple method seems effective in allowing sophisticated manipulations of high-level abstract concepts, enabling users to explore, with little effort, how the model would respond in different scenarios.

We believe this work provides a step forward towards making neural networks more interpretable by addressing one of the natural ways in which humans attempt to understand complex models – by imagining hypothetical scenarios and thinking about how they would impact the model. By providing a method that allows end-users to define their own meaningful concepts and to test such hypothetical scenarios, we seek to answer the necessity that humans have of validating their expectations regarding how a model works. Ultimately, we hope that this method contributes to promoting better testing and validation of artificial neural networks, improving the trust that humans might have in these systems.

In the future, we are interested in exploring how to leverage this method to search for learned biases and identify missing or undesired associations between concepts in artificial neural networks.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] Y. Zhang, P. Tiňo, A. Leonardis, K. Tang, A survey on neural network interpretability, IEEE Trans. Emerg. Top. Comput. Intell. 5 (2021) 726–742.

[2] P. Hitzler, F. Bianchi, M. Ebrahimi, M. K. Sarker, Neural-symbolic Integration and the Semantic Web, Semantic Web 11 (2020) 3–11.

[3] M. D. Zeiler, R. Fergus, Visualizing and Understanding Convolutional Networks, in: ECCV'14, 2014.

[4] M. Sundararajan, A. Taly, Q. Yan, Axiomatic attribution for deep networks, in: ICML'17, 2017.

[5] M. Ancona, E. Ceolini, C. Öztireli, M. Gross, Towards better understanding of gradient-based attribution methods for Deep Neural Networks, in: ICLR'18, 2018.

[6] A. Ignatiev, N. Narodytska, J. Marques-Silva, Abduction-based explanations for machine learning models, in: AAAI'19, 2019.
[7] S. Rebuffi, R. Fong, X. Ji, A. Vedaldi, There and Back Again: Revisiting Backpropagation Saliency Methods, in: CVPR'20, 2020.
[8] M. Ivanovs, R. Kadikis, K. Ozols, Perturbation-based methods for explaining deep neural networks: A survey, Pattern Recognit. Lett. 150 (2021) 228–234.
[9] G. P. J. Schmitz, C. Aldrich, F. S. Gouws, ANN-DT: an algorithm for extraction of decision trees from artificial neural networks, IEEE Trans. Neural Networks 10 (1999) 1392–1401.
[10] M. G. Augasta, T. Kathirvalavakumar, Reverse engineering the neural networks for rule extraction in classification problems, Neural Process. Lett. 35 (2012) 131–150.
[11] M. T. Ribeiro, S. Singh, C. Guestrin, "why should I trust you?": Explaining the predictions of any classifier, in: SIGKDD'16, 2016.
[12] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A Survey of Methods for Explaining Black Box Models, ACM Comput. Surv. 51 (2019) 93:1–93:42.
[13] F. Shakerin, G. Gupta, Induction of non-monotonic logic programs to explain boosted tree models using LIME, in: AAAI'19, 2019.
[14] F. Shakerin, G. Gupta, White-box induction from SVM models: Explainable AI with logic programming, Theory Pract. Log. Program. 20 (2020) 656–670.
[15] J. Adebayo, M. Muelly, I. Liccardi, B. Kim, Debugging Tests for Model Explanations, in: NeurIPS'20, 2020.
[16] E. Chu, D. Roy, J. Andreas, Are Visual Explanations Useful? A Case Study in Model-in-the-Loop Prediction, CoRR abs/2007.12248 (2020).
[17] H. Shen, T. K. Huang, How Useful Are the Machine-Generated Interpretations to General Users? A Human Evaluation on Guessing the Incorrectly Predicted Labels, in: HCOMP'20, 2020.
[18] Y. Belinkov, Probing Classifiers: Promises, Shortcomings, and Advances, Comput. Linguistics 48 (2022) 207–219.
[19] R. Q. Quiroga, L. Reddy, G. Kreiman, C. Koch, I. Fried, Invariant visual representation by single neurons in the human brain, Nature 435 (2005) 1102–1107.
[20] R. Q. Quiroga, Concept cells: the building blocks of declarative memory functions, Nature Reviews Neuroscience 13 (2012) 587–597.
[21] E. Poeta, G. Ciravegna, E. Pastor, T. Cerquitelli, E. Baralis, Concept-based explainable artificial intelligence: A survey, CoRR abs/2312.12936 (2023).
[22] G. Schwalbe, B. Finzel, A comprehensive taxonomy for explainable artificial intelligence: a systematic survey of surveys on methods and concepts, Data Min. Knowl. Discov. 38 (2024) 3043–3101.
[23] B. Kim, M. Wattenberg, J. Gilmer, C. J. Cai, J. Wexler, F. B. Viégas, R. Sayres, Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV), in: ICML'18, 2018.
[24] Z. Chen, Y. Bei, C. Rudin, Concept whitening for interpretable image recognition, Nature Machine Intelligence 2 (2020) 772–782.
[25] M. de Sousa Ribeiro, J. Leite, Aligning Artificial Neural Networks and Ontologies towards Explainable AI, in: AAAI'21, 2021.
[26] T. Miller, Explanation in Artificial Intelligence: Insights from the Social Sciences, Artif. Intell. 267 (2019) 1–38.
[27] R. M. J. Byrne, Counterfactuals in explainable artificial intelligence (XAI): evidence from human reasoning, in: IJCAI'19, 2019.
[28] R. Guidotti, Counterfactual explanations and how to find them: literature review and benchmarking, Data Mining and Knowledge Discovery (2022).
[29] F. Yang, N. Liu, M. Du, X. Hu, Generative Counterfactuals for Neural Networks via Attribute-Informed Perturbation, SIGKDD Explor. 23 (2021) 59–68.
[30] I. Stepin, J. M. Alonso, A. Catalá, M. Pereira-Fariña, A Survey of Contrastive and Counterfactual Explanation Generation Methods for Explainable Artificial Intelligence, IEEE Access 9 (2021)

11974–12001.
[31] F. Hvilshøj, A. Iosifidis, I. Assent, ECINN: Efficient Counterfactuals from Invertible Neural Networks, in: BMVC'21, 2021.
[32] G. Goh, N. Cammarata, C. Voss, S. Carter, M. Petrov, L. Schubert, A. Radford, C. Olah, Multimodal Neurons in Artificial Neural Networks, Distill (2021).
[33] T. Okuyama, Social memory engram in the hippocampus, Neuroscience Research 129 (2018) 17–23.
[34] M. de Sousa Ribeiro, L. Krippahl, J. Leite, Explainable Abstract Trains Dataset, CoRR abs/2012.12115 (2020).
[35] J. Larson, R. S. Michalski, Inductive inference of VL decision rules, SIGART Newsl. 63 (1977) 38–44.
[36] J. Ferreira, M. de Sousa Ribeiro, R. Gonçalves, J. Leite, Looking Inside the Black-Box: Logic-based Explanations for Neural Networks, in: KR'22, 2022.
[37] M. Tucker, P. Qian, R. Levy, What if This Modified That? Syntactic Interventions with Counterfactual Embeddings, in: ACL/IJCNLP'21, 2021.
[38] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, C. Igel, Detection of traffic signs in real-world images: The german traffic sign detection benchmark, in: IJCNN'13, 2013.
[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, International Journal of Computer Vision (IJCV) 115 (2015) 211–252.
[40] U. UNECE, Convention on Road Traffic of 1968 and European Agreement Supplementing the Convention (2006 Consolidated Versions), United Nations Publications, 2007.
[41] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, L. Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks, in: CVPR'18, 2018.
[42] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. A. Specter, L. Kagal, Explaining Explanations: An Overview of Interpretability of Machine Learning, in: DSAA'18, 2018.
[43] M. K. Sarker, N. Xie, D. Doran, M. L. Raymer, P. Hitzler, Explaining trained neural networks with semantic web technologies: First steps, in: NeSy'17, 2017.
[44] X. Li, Y. Shi, H. Li, W. Bai, C. C. Cao, L. Chen, An Experimental Study of Quantitative Evaluations on Saliency Methods, in: KDD'21, 2021.
[45] Y. Elazar, S. Ravfogel, A. Jacovi, Y. Goldberg, Amnesic probing: Behavioral explanation with amnesic counterfactuals, Trans. Assoc. Comput. Linguistics 9 (2021) 160–175.
[46] A. Ghorbani, J. Wexler, J. Y. Zou, B. Kim, Towards automatic concept-based explanations, in: NeurIPS'19, 2019.
[47] B. Zhou, D. Bau, A. Oliva, A. Torralba, Interpreting deep visual representations via network dissection, IEEE Trans. Pattern Anal. Mach. Intell. 41 (2019) 2131–2145.
[48] V. A. C. Horta, I. Tiddi, S. Little, A. Mileo, Extracting knowledge from deep neural networks through graph analysis, Future Gener. Comput. Syst. 120 (2021) 109–118.
[49] J. Jo, Y. Bengio, Measuring the tendency of cnns to learn surface statistical regularities abs/1711.11561 (2017).
[50] K. Meng, D. Bau, A. Andonian, Y. Belinkov, Locating and editing factual associations in GPT, in: NeurIPS'22, 2022.
[51] L. Bereska, S. Gavves, Mechanistic interpretability for AI safety - A review, Trans. Mach. Learn. Res. (2024).
[52] P. Henriksen, F. Leofante, A. Lomuscio, Repairing misclassifications in neural networks using limited data, in: SAC'22, 2022.
[53] D. L. Calsi, M. Duran, T. Laurent, X. Zhang, P. Arcaini, F. Ishikawa, Adaptive search-based repair of deep neural networks, in: GECCO'23, 2023.
[54] J. Kim, N. Humbatova, G. Jahangirova, P. Tonella, S. Yoo, Repairing DNN architecture: Are we there yet?, in: ICST'23, 2023.
[55] N. D. Cao, W. Aziz, I. Titov, Editing factual knowledge in language models, in: EMNLP'21, 2021.