Feature-Guided Analysis of Neural Networks

Divya Gopinath[1], Luca Lungeanu[2], Ravi Mangal[3], Corina Păsăreanu[1,3], Siqi Xie[3], and Huanfeng Yu[4]

[1] KBR, NASA Ames, Moffett Field CA 94035, USA
[2] Lynbrook High School, San Jose CA, 95129, USA
[3] Carnegie Mellon University, Pittsburgh PA 15213, USA
[4] Boeing Research & Technology, Santa Clara CA, USA

Abstract. Applying standard software engineering practices to neural networks is challenging due to the lack of high-level abstractions describing a neural network's behavior. To address this challenge, we propose to extract high-level task-specific *features* from the neural network internal representation, based on monitoring the neural network activations. The extracted feature representations can serve as a link to high-level requirements and can be leveraged to enable fundamental software engineering activities, such as automated testing, debugging, requirements analysis, and formal verification, leading to better engineering of neural networks. Using two case studies, we present initial empirical evidence demonstrating the feasibility of our ideas.

Keywords: Features, Neural Networks, Software Engineering

1 Introduction

The remarkable computational capabilities unlocked by neural networks have led to the emergence of a rapidly growing class of neural-network based software applications. Unlike traditional software applications whose logic is driven from input-output specifications, neural networks are inherently *opaque*, as their logic is learned from examples of input-output pairs. The lack of high-level abstractions makes it challenging to interpret the logical reasoning employed by a neural network and hinders the use of standard software engineering practices such as automated testing, debugging, requirements analysis, and formal verification that have been established for producing high-quality software.

In this work, we aim to address this challenge by proposing a *feature-guided* approach to neural network engineering. Our proposed approach is illustrated in Figure 1. We draw from the insight that, in a neural network, early layers typically extract the important features of the inputs and the dense layers close to the output contain logic in terms of these features to make decisions [12]. The approach therefore first extracts high-level, human-understandable feature representations from the trained neural network which allows us to formally link domain-specific, human-understandable features to the internal logic of a trained model. This in turn enables us to reason about the model through the lens of the features and to drive the above mentioned software engineering activities.

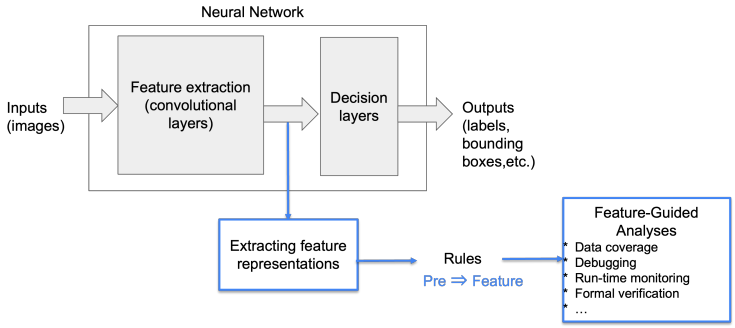

Fig. 1: Proposed Approach

For feature representations, we seek to extract associations between the activation values at the intermediate layers and higher-level abstractions that have clear semantic meaning (e.g., objects in a scene or weather conditions). We present an algorithm to extract these high-level feature representations in the form of *rules* (`pre` $\implies$ `post`) where the precondition (`pre`) is a box over the latent space at an internal layer and the postcondition (`post`) denotes the presence (or absence) of the feature.

The formal, checkable rules enable us to evaluate the quality of the datasets, retrieve and label new data, understand scenarios where models make correct and incorrect predictions, detect incorrect (or out-of-distribution) samples at run-time, and verify models against human-understandable requirements.

We evaluate our algorithm for extracting feature representations and the downstream analyses using two networks trained for computer vision tasks, namely TaxiNet [4, 9], a regression model for center-line tracking on airport runways, and YOLOv4-Tiny [14], an object detection model trained on the nuImages [6] dataset for autonomous driving.

2 Extracting Feature Representations

Algorithm 2.1 describes the method for extracting the representation of a particular feature from a trained neural network. A feed-forward neural network $f : \mathbb{R}^n \to \mathbb{R}^m$ is organized in multiple *layers*, each consisting of computational units called *neurons*. Each neuron takes a weighted sum of the outputs from the previous layer and applies a non-linear *activation function* on it. The algorithm requires a *small* dataset D where each raw input is labeled with 0 or 1 indicating whether the feature under consideration is absent or present. The algorithm takes as inputs a neural network f, the dataset D, the index l of the layer used for extracting the feature representations. The first step of the algorithm (line 2) is to construct a new dataset A where each raw input x is replaced by the corresponding activation value a output by layer l ($f^l(x)$ denotes the output of f at layer l for input x). Next, the algorithm invokes a learning procedure to learn a classifier r that separates activation values that map to feature being present from activation values that map to feature absence (line 3).

Algorithm 2.1: Extracting Feature Representations

Inputs: A neural classifier $f \in \mathbb{R}^n \to \mathbb{R}^m$, dataset $D \subseteq \mathbb{R}^n \times \{0,1\}$, $|D| = N$, and layer $l \in \{1, \ldots, k-1\}$, where k is the number of layers in f
Output: Representation r for the feature

```
1 FeatRep(f, D, l):
2     A := {(a, y) | (x, y) ∈ D ∧ a = f^l(x)} //f^l is the output of f at layer l
3     r := Learn(A)
4     return r
```

We use decision tree learning on line 3 to extract feature representations as a set of rules of the form $\texttt{pre} \Rightarrow \{0,1\}$; `pre` in each rule is a condition on neuron values at layer l, and 0 or 1 indicates whether the rule corresponds to the feature being absent or present. `pre` is a box in the activation space of layer l, i.e., $\bigwedge_{N_j \in \mathcal{N}_l}(N_j(x) \in [v_j^L, v_j^U])$. Here $\mathcal{N}_l$ is the set of neurons at layer l, and v_j^L and v_j^U are lower and upper bounds for the output of neuron N_j. The rules mined by decision-tree learning partition the activation space at a given inner layer. Some partitions may be impure containing inputs both with and without the feature. We only select pure rules, having 100% precision on d. We return these rules as r. Note that there can be activation values for which no rule in r is satisfied and we are unable to say whether the feature is absent or present.

3 Feature-Guided Analyses

The extracted feature representations as formal, checkable rules enable multiple analyses, as listed below.

- **Data analysis and testing.** We can assess the quality of the training and testing data in terms of coverage of different features. We can leverage the extracted feature representations to automatically retrieve new data that has the necessary features, by checking that the (unlabeled) data satisfies the corresponding rules. We can also use the extracted rules to label new data with their corresponding features, enabling further data-coverage analysis.
- **Debugging and explanations of network behavior.** We can leverage the feature rules to uncover the high-level, human-understandable reasons for a neural network model making correct and incorrect predictions. In the latter case we can repair the model, which involves automatically selecting and training based on inputs with features that caused incorrect predictions.
- **Formulation and analysis of requirements.** Extracted feature representations are the key to enabling verification of models with respect to high-level safety requirements ($\texttt{pre} \implies \texttt{post}$). Specifically, the constraint `pre` in the requirement expressed over features can be translated into a constraint $\texttt{pre}'$ expressed over activation values, by substituting the features with their corresponding representations. The modified requirement $\texttt{pre}' \implies \texttt{post}$ can be checked automatically using off-the-shelf verification tools [10].

- **Run-time monitoring.** We can also enforce safety properties at run-time. For instance, we can use `pre′` as above to check (at run-time) whether inputs satisfy a desired precondition, and reject the ones that don't.
- **Conformance with the operational design domain (ODD).** This is a particular instance of the case above, where we use the rules to formally capture the model's expected domain of operation and use a run-time guard to ensure that the model is not used in scenarios outside its ODD. A related problem is out-of-distribution detection, where we can similarly formulate the conditions under which the model is not supposed to operate and use run-time monitoring to enforce it.

One can also check overlap between feature rules, using off-the-shelf decision procedures, to uncover spurious correlations between the different features that are learned by the network. We envision many other applications for these rules, whose exploration we leave for the future.

4 Case Studies

We use two case studies to present initial empirical evidence in support of our ideas. In particular, we show that Algorithm 2.1 with decision tree learning is successful in extracting feature representations. We also demonstrate how these representations can be used for analyzing the behavior of neural networks.

4.1 Center-line Tracking with TaxiNet

We first analyzed TaxiNet, a perception model for center-line tracking on airport runways [4, 9]. It takes runway images as input and produces two outputs, cross-track (CTE) and heading angle (HE) errors which indicate the lateral and angular distance respectively of the nose of the plane from the center-line of the runway. We analyzed a CNN model provided by our industry partner, with 24 layers including three dense layers (100/50/10 neurons) before the output layer. It is critical that the TaxiNet model functions correctly and keeps the plane safe without running off the taxiway. The domain experts provided a specification for correct output behavior: $|y_0 - y_{0ideal}| \leq 1.0m \wedge |y_1 - y_{1ideal}| \leq 5$ *degrees*. One can evaluate the model correctness using Mean Absolute Error (MAE) on a test set (CTE:0.366, HE:1.645).

Feature Elicitation We first need to identify the high-level features that are relevant for the task. These could be some of the simulator parameters (for images generated from a simulator) and/or could be derived from high-level system (natural language) requirements. This is a challenging process requiring several iterations in collaboration with the domain experts. We obtained a list of 10 features: center-line, shadow, skid, position, heading, time-of-day, weather, visibility, intersection (junction) and objects (runway lights, birds, etc.) and values of interest for each feature respectively.

Data Analysis and Annotations We manually annotated a subset of 450 images from the test set with values for each feature. An initial data-coverage

Table 1: Rules for TaxiNet: d: annotated dataset, $\#d$: total number of instances for that feature value in d, R_d: recall (%) on d, P_v,R_v: precision (%) and recall (%) on validation set. Rules with highest R_d are shown.

Feature	Rule	Metrics			
		#d	R_d	P_v	R_v
Center-line	$N_{3,9} <= 1.39 \wedge N_{3,9} > -0.98 \wedge N_{3,1} > -0.99$ $\wedge N_{3,4} > -0.99 \implies present$	202	92	93	100
	$N_{3,9} > 1.39 \wedge N_{3,6} <= -0.81 \implies absent$	25	40	100	12
Shadow	$N_{2,45} <= -0.75 \wedge N_{1,50} > -0.91$ $\wedge N_{1,9} <= -0.95 \implies present$	30	86	100	69.23
	$N_{2,4} <= -0.73 \wedge N_{2,3} > 0.06 \wedge N_{2,9} > -0.98 \implies absent$	200	94.5	97	100
Skid	$N_{2,8} <= -0.98 \wedge N_{2,10} <= 0.32 \implies dark$	40	52.5	94.44	43.5
	$N_{1,28} <= -0.93 \wedge N_{2,58} > -0.88 \implies no$	5	60	0	0
	$N_{2,8} > -0.997 \wedge N_{2,48} > -0.991 \wedge N_{2,42} <= -0.342$ $\wedge N_{2,25} <= 1.82 \implies light$	182	97.8	93.4	95
Position	$N_{2,2} > -0.99 \wedge N_{2,24} <= -0.3 \wedge N_{2,9} <= -1.19 \implies right$	101	90	92.3	95.09
	$N_{1,26} > -0.55 \wedge N_{1,20} <= -0.29 \wedge N_{1,52} <= -0.96 \implies left$	109	91	100	75.22
	$N_{3,6} > -0.17 \wedge N_{3,6} <= 0.45 \wedge N_{3,3} > -0.38 \wedge N_{3,7} <= -0.55$ $\wedge N_{3,0} <= 2.56 \wedge N_{3,5} <= -0.95 \implies on$	11	45	13.5	45.45
Heading	$N_{1,5} > 3.29 \wedge N_{1,90} <= -0.87 \wedge N_{1,81} <= -0.76 \implies away$	120	65	62.2	90.6
	$N_{1,5} <= 3.29 \wedge N_{1,37} > -0.84 \wedge N_{1,50} <= 8.22 \wedge N_{1,53} <= -0.39$ $\wedge N_{1,64} > -0.98 \wedge N_{1,45} <= -0.26 \wedge N_{1,34} <= 12.21 \implies towards$	102	83	73.9	16.5

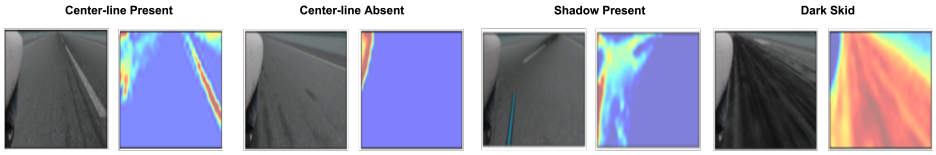

Fig. 2: Images satisfying rules for features

analysis of the distribution of the values for every feature across all the images, revealed many gaps. For instance, there were only day-time images, with only cloudy weather and all the images had high visibility. Also apart from runway lights, there were no images with any other objects on the runway. The analysis proved already useful, providing feedback to the experts with regard to the type of images that need to be added to improve the training and testing of the model.

Extracting Feature Rules We invoke Algorithm 2.1 to obtain rules in terms of the values of the neurons at the three dense layers of the network. Note that for each feature, we mined a separate rule for every value of interest. We used half of the annotated set of 450 images for extraction (d in Algorithm 2.1) and the remaining for validation of the rules. There are multiple rules extracted for each feature; each rule is associated with a support value (# of instances in d satisfying the rule) and has 100% precision on them since we only extract pure rules. The results are summarized in Table 1, indicating some high-quality rules (for "center-line present" , "shadow present" , "light skid", "position left", "position right"), measured on the validation set.

Figure 2 displays some of the images satisfying different rules. The corresponding heat maps were created by computing the image pixels impacting the neurons in the feature rule [7]. Note that for the "center-line present" rule, the part of the

image impacting the rule (highlighted in red) is the center-line, indicating that indeed the rules identify the feature. On the other hand, in the absence of the center-line, it is unclear what information is used by the model (and the image leads to error). The heatmaps for the shadow and skid also correctly highlight the part of the image with the shadow of the nose and the skid marks. We used such visualization techniques to further validate the rules.

Labeling New Data The rules extracted based on a small set of manually annotated data can be leveraged to annotate a much larger data set. We used the rules for center-line (present/absent) to label all of the test data (2000 images). We chose the rule with highest R_d for the experiments. However, more rules could be chosen to increase coverage. 1822 of the images satisfied the rule for "center-line present" and 79 images for "center-line absent". We visually checked some of the images to estimate the accuracy of the labelling. We similarly annotated more images for the shadow and skid features. These new labels enable further data-coverage analysis over the train and test datasets.

Feature-Guided Analysis We performed preliminary experiments to demonstrate the potential of feature-guided analyses. We first calculated the model accuracy (MAE) on subsets of the data labelled with the feature present and absent respectively. We also determined the % of inputs in the respective subsets violating the correctness property. The results are summarized in Table 2.

Table 2: Feature-Guided Analysis Results

Rule	**MAE CTE**	**MAE HE**	errors
"center-line present"	0.36	1.63	45%
"center-line absent"	0.62	2.68	75%
"shadow present"	0.66	2.23	42%
"shadow absent"	0.34	1.55	7%
"dark skid"	0.43	1.84	52%
"light or no skid"	0.33	1.49	42%

These results can be used by developers to better understand and debug the model behavior. For instance, the model accuracy computed for the subsets with "shadow present" and "dark skid", respectively, is poor and also a high % of the respective inputs violate the correctness property. This information can be used by developers to retrieve more images with shadows and dark skids, to retrain the model and improve its performance. The extracted rules can be leveraged to automate the retrieval.

Furthermore, we observe that in the absence of the center-line feature, the model has difficulty in making correct predictions. This is not surprising, as the presence of the center-line can be considered as a (rudimentary) input requirement for the center-line tracking application. Indeed, in the absence of the center-line it is hard to envision how the network can estimate correctly the airplane position from it. The network may use other clues on the runway, leading to errors. We can thus consider the presence of the center-line feature as part of the ODD for the application. The rules for the center-line feature can be deployed as a *run-time monitor* to either pass inputs satisfying the rules for "present" or reject those that satisfy the rules for "absent", ensuring that the model operates in the safe zone as defined by the ODD, and at the same time increasing its accuracy.

We also experimented with generating rules to explain correct and incorrect behavior in terms of *combinations* of features such as: $(center-line\ present) \wedge (shadow\ absent) \wedge (on\ position) \implies correct$, and $\neg(center-line\ present) \wedge$

Table 3: Rules for YOLOv4-Tiny (same metrics as in Table 1).

Feature	Rule	Metrics			
		#**d**	R_d	P_v	R_v
pedestrian.moving	pre(21 *terms*) $\implies$ *present*	1110	48	72	29
	pre(15 *terms*) $\implies$ *absent*	894	40	74	29
vehicle.parked	pre(10 *terms*) $\implies$ *present*	890	25	71	20
	pre(19 *terms*) $\implies$ *absent*	1114	43	70	32
pedestrian	pre(25 *terms*) $\implies$ *present*	1375	57	70	35
	pre(14 *terms*) $\implies$ *absent*	629	41	77	22
vehicle	pre(20 *terms*) $\implies$ *present*	1616	75	91	59
	pre(11 *terms*) $\implies$ *absent*	388	50	69	31

$(\textit{heading away}) \wedge (\textit{position right}) \implies \textit{incorrect}$. [1]. These rules could be further used by developers to better understand and debug the model behavior.

4.2 Object Detection with YOLOv4-Tiny

We conducted another case study with a more challenging network, an object detector, to evaluate the quality of the extracted feature representations. For this study, we use the nuImages dataset, a public large-scale dataset for autonomous driving [1, 6]. It contains 93000 images collected while driving around in actual cities. To facilitate computer vision tasks such as object detection for autonomous driving, each image comes labeled with 2d bounding boxes and the corresponding object labels (from one of 23 object classes). Each labeled object also comes with additional attribute annotations. For instance, the objects labeled *vehicle* carry additional annotations like *vehicle.moving*, *vehicle.stopped*, and *vehicle.parked*. Overall, the dataset has 12 categories of additional attribute annotations. We trained a YOLOv4-Tiny object detection model [14, 2] on this dataset. YOLOv4-Tiny has 37 layers with 21 convolutional layers and 2 YOLO layers.

We leveraged the attribute annotations associated with each object as the feature labels (thus no manual labeling was necessary). For extracting feature representations, we run Algorithm 2.1 on a subset of 2000 images from the nuImages dataset, and then evaluate the extracted representations on a separate validation set of 2000 images.

Table 3 describes our results. We used layer 28 of the YOLOv4-Tiny model to extract the feature representations. For brevity, we only report the number of terms in the rule precondition, i.e., the number of neurons that appeared in the constraints, instead of describing the exact rule in Table 3. Note that layer 28 has 798720 neurons. Strikingly, the extracted rules only have between 10 to 25 terms in their preconditions, and yet achieve precision (P_v) between $69 - 74\%$. The recall (R_v) values are also encouraging, and can be improved further by considering more than one rule for each feature value (here, we only consider pure rules with the highest recall R_d on dataset d used for feature extraction).

[1] The procedure to generate these rules has been omitted for brevity.

4.3 Challenges and Mitigations

Identifying relevant features is non-trivial and requires refinement and extensive discussions with domain experts. The feature annotations may need to be provided manually which is expensive and error-prone. However, we only need a small annotated dataset to extract the representations, which can be used to further annotate unlabeled data. The extracted rules may be incorrect (e.g., due to unbalanced annotated data). We mitigate by carefully validating them using a separate validation set and visualization techniques. It could also be that the network did not learn some important features. To address the issue, in future work, we plan to investigate neuro-symbolic approaches to build networks that are aware of high-level features and satisfy (by construction) the safety requirements.

5 Related Work

There is growing interest in developing software engineering approaches for machine learning in general, and neural networks specially, investigating requirements for neural networks [3], automated testing [16], debugging and fault localization [8], to name a few. Our work contributes with a feature-centric view of neural network behavior that links high-level requirements with the internal logic of the trained models to enable better testing and analysis of neural networks.

A closely related work [18] uses high-level features to guide neural network analysis. However, the features are extracted from input images, not from the internal neural network representation. Further, the work only considers testing, not other software engineering activities.

Our work is also related to concept analysis [17, 11, 15, 13] which seeks to develop explanations of deep neural network behavior in terms of concepts specified by users. We propose to use high-level features for multiple software engineering activities, which go beyond explanations. Moreover, the use of decision tree learning makes our representations relatively cheap to extract. Note that there are other works that use decision tree learning to distill neural network input-output behavior, e.g., [5]; however none of them extract high-level features from the network's internal representation.

6 Conclusion

We proposed to extract high-level feature representations related to domain-specific requirements to enable analysis and explanation of neural network behavior. We presented initial empirical evidence in support of our ideas. In future work, we plan to further investigate meaningful requirements for neural networks and effective techniques for checking them. We also plan to apply Marabou [10] for the verification of safety properties expressed in terms of high-level features. Finally, we plan to investigate neuro-symbolic techniques to develope high-assurance neural network models.

References

1. nuimages, https://www.nuscenes.org/nuimages
2. Yolov4-tiny, https://github.com/WongKinYiu/PyTorch_YOLOv4
3. Ashmore, R., Calinescu, R., Paterson, C.: Assuring the machine learning lifecycle: Desiderata, methods, and challenges. ACM Comput. Surv. **54**(5), 111:1–111:39 (2021). https://doi.org/10.1145/3453444, https://doi.org/10.1145/3453444
4. Beland, S., Chang, I., Chen, A., Moser, M., Paunicka, J.L., Stuart, D., Vian, J., Westover, C., Yu, H.: Towards assurance evaluation of autonomous systems. In: IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020. pp. 84:1–84:6. IEEE (2020). https://doi.org/10.1145/3400302.3415785, https://doi.org/10.1145/3400302.3415785
5. Bondarenko, A., Aleksejeva, L., Jumutc, V., Borisov, A.: Classification tree extraction from trained artificial neural networks. Procedia Computer Science **104**, 556–563 (2017). https://doi.org/https://doi.org/10.1016/j.procs.2017.01.172, https://www.sciencedirect.com/science/article/pii/S1877050917301734, iCTE 2016, Riga Technical University, Latvia
6. Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: nuscenes: A multimodal dataset for autonomous driving. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11621–11631 (2020)
7. Chattopadhay, A., Sarkar, A., Howlader, P., Balasubramanian, V.N.: Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 839–847. IEEE (2018)
8. Fahmy, H.M., Pastore, F., Briand, L.C.: HUDD: A tool to debug dnns for safety analysis. In: 44th 2022 IEEE/ACM International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2022, Pittsburgh, PA, USA, May 22-24, 2022. pp. 100–104. IEEE (2022). https://doi.org/10.1109/ICSE-Companion55297.2022.9793750, https://doi.org/10.1109/ICSE-Companion55297.2022.9793750
9. Frew, E., McGee, T., Kim, Z., Xiao, X., Jackson, S., Morimoto, M., Rathinam, S., Padial, J., Sengupta, R.: Vision-based road-following using a small autonomous aircraft. In: 2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720). vol. 5, pp. 3006–3015 Vol.5 (2004). https://doi.org/10.1109/AERO.2004.1368106
10. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)
11. Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., sayres, R.: Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 2668–2677. PMLR (10–15 Jul 2018), https://proceedings.mlr.press/v80/kim18d.html
12. Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., Carter, S.: Zoom in: An introduction to circuits. Distill (2020). https://doi.org/10.23915/distill.00024.001, https://distill.pub/2020/circuits/zoom-in
13. Schwalbe, G.: Concept embedding analysis: A review (2022). https://doi.org/10.48550/ARXIV.2203.13909, https://arxiv.org/abs/2203.13909

14. Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M.: Scaled-yolov4: Scaling cross stage partial network. In: Proceedings of the IEEE/cvf conference on computer vision and pattern recognition. pp. 13029–13038 (2021)
15. Yeh, C.K., Kim, B., Ravikumar, P.: Human-centered concept explanations for neural networks. In: Neuro-Symbolic Artificial Intelligence: The State of the Art, pp. 337–352. IOS Press (2021)
16. Zhang, M., Zhang, Y., Zhang, L., Liu, C., Khurshid, S.: Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In: Huchard, M., Kästner, C., Fraser, G. (eds.) Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018. pp. 132–142. ACM (2018). https://doi.org/10.1145/3238147.3238187, https://doi.org/10.1145/3238147.3238187
17. Zhou, B., Sun, Y., Bau, D., Torralba, A.: Interpretable basis decomposition for visual explanation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 119–134 (2018)
18. Zohdinasab, T., Riccio, V., Gambi, A., Tonella, P.: Efficient and effective feature space exploration for testing deep learning systems. ACM Trans. Softw. Eng. Methodol. (jun 2022). https://doi.org/10.1145/3544792, https://doi.org/10.1145/3544792, just Accepted