Gaussian-based runtime detection of out-of-distribution inputs for neural networks *

Vahid Hashemi[2 [0000−0002−9167−7417]], Jan Křetínský[1 [0000−0002−8122−2881]], Stefanie Mohr[1 [0000−0002−8630−3218]], and Emmanouil Seferis[1,2]

[1] Technical University of Munich, Germany
[2] AUDI AG, Ettingerstr. 60, 85057 Ingolstadt, Germany

Abstract. In this short paper, we introduce a simple approach for runtime monitoring of deep neural networks and show how to use it for out-of-distribution detection. The approach is based on inferring Gaussian models of some of the neurons and layers. Despite its simplicity, it performs better than recently introduced approaches based on interval abstractions which are traditionally used in verification.

1 Introduction

Learning deep neural networks (DNN) [2] has shown remarkable success in practically solving a large number of hard and previously intractable problems. However, direct applications in safety-critical domains, such as automated driving, are hindered by the lack of practical methods to *guarantee their safety*, e.g. [3,4]. This poses a serious problem for industrial adoption of DNN-based systems. Companies struggle to comply with safety regulations such as SOTIF [42], both due to lack of techniques to demonstrate safety in the presence of DNN as well as due to the actual lack of safety, e.g. accidents in automated cars due to errors in DNN-based perception system used [6].

One of the key requirements is the ability to detect novel inputs [43], for which the DNN has not been trained and thus the only responsible answer is "don't know". Such inputs are also called *out-of-distribution (OOD) examples* [20]. Whenever such inputs occur, an alarm should be raised announcing the unreliability of the current output of the DNN, so that rectifying actions can be taken. Various runtime monitors for this task have already been proposed recently. Cheng et al. [1] monitor which subsets of neurons in a given layer are activated for known inputs; whenever a very different subset is activated, an alarm is raised. Henzinger et al. [39] monitor activation values of neurons and envelop the tuples into hyper-boxes (multidimensional intervals) along the program analysis tradition; whenever a very different tuple is observed (outside of the boxes), an alarm is raised.

In this short paper, we propose a very light-weight and scalable approach. Similarly to [39], we monitor the activation values. However, instead of discrete and exact

* This research was funded in part by the DFG research training group *CONVEY* (GRK 2428), the DFG project 383882557 - Statistical Unbounded Verification (KR 4890/2-1), the project *Audi Verifiable AI*, and the BMWi funded *KARLI* project (grant 19A21031C).

enveloping, we learn a more continuous and fuzzy representation of the recorded experienece, namely a Gaussian model of each monitored neuron. Whenever many neurons have sufficiently improbable activation values on the current input, we raise an alarm. Surprisingly, our simple monitor is equally or even more accurate than the similar state-of-the-art [39] even though we take no correlation of the activation values of different neurons into account and instead we monitor each of the neurons separately, in contrast to the multi-dimensional boxes of [39].

Our contribution can be summarized as follows:

- We present a new and simple method for OOD detection based on Gaussian models of neuron activation values.
- We show that our method performs better than state-of-the-art techniques for out-of-distribution (OOD) detection.

Related Work In our work, we focus on the detection of OOD-inputs, arguably [20] one of the major problems in AI safety.

State of the art A recent work by Henzinger et al. [39] is very similar to our approach. The authors consider the neuron activations of one layer for all samples of the training data. For each class in the dataset, they collect the activation vectors of the class samples, and cluster them using k-Means [40]. They increase the number of clusters successively, until the relative improvement drops below a given threshold τ. For each cluster, they construct a box abstraction that contains all samples of that cluster. In the end, each class in the data corresponds to a set of boxes. Finally, during testing, they check whether the activation vector of a new sample is contained in one of the boxes of its predicted class; if not, they raise an alarm. This approach can be extended to more layers, by taking the element-wise boolean AND of the layer "decisions". That is, an input is accepted if only if it is contained in the abstractions of all monitored layers. While the idea of looking at the activations of neurons in a layer is similar to our approach, the difference is in the detection of OOD samples. In contrast to using box-abstractions, we use Gaussian models. This reflects better the actual distribution of values of the neurons, as can be seen in Section 4.2.

OOD-Detection Previous works have suggested, for example, using the maximum class probability or the entropy of the predicted class distribution as an OOD indicator [21], or training a classifier to distinguish clean and perturbed data, using ensembles of classifiers trained on random shuffles of the training data [22]. Besides, two popular approaches closely resemble the methods of runtime monitoring, namely ODIN [23] and the Mahalanobis-based detector [24]. ODIN first applies temperature scaling on the softmax outputs of a DNN to reduce the standard DNN overconfidence, and then applies a small adversarial–like perturbation of the input. If after that the maximum class score is below some threshold, the sample is considered to be OOD.

In contrast, the detector of [24] measures the probability density of a test sample by using a distance-based classifier. Another line of work involves generative models

for OOD detection, attempting to model the distribution of the data, such as in [25]. By definition, OOD detection runs at test time, and thus many proposed approaches can be viewed under this setting. Other related approaches include using Bayesian learning methods [15], which can output prediction uncertainties, DNN testing [3], which are methods attempting to find problematic inputs, or building DNN architectures that are robust by construction, for example using interval bound propagation, abstract interpretation, or other methods [12,13,14].

2 Preliminaries

2.1 Deep Neural Networks

DNNs come in various architectures suitable for different tasks, however, at the core, they are composed of multiple *layers* of computation units called *neurons*. The task of a neuron is to read an input, calculate a weighted sum, apply a function called the *activation function* on it and output the result, called the *activation value*. We number the layers $1, 2, \ldots, L$ where layer 1 is called the input layer, layers $2, \ldots (L-1)$ are called the hidden layers and layer L is called the output layer.

More formally, given an input $\vec{x}$ to the DNN, we have:

$$\vec{h}^1 = \vec{x}$$
$$\vec{h}^{l+1} = \vec{\phi}^{l+1}(\vec{h}^l) \;\; l = 2, ..., L$$

where $\phi_i^l(\vec{x})$ defines the element-wise computation of the neurons $i = 1, ..., N_l$ in layer l. The details of the computation are not necessary to understand the following work.

DNN can perform various tasks, the most usual being classification and regression. Whereas the first type labels its input with a category from a finite subset of classes, the second type outputs non discrete but real values. We consider only classification DNNs in this work. Neuron activations are vectors of activation values produced by neurons in some layer of a DNN. It is generally believed that layers closer to the output encode more complex features. This result has been supported by our results, which can be seen in Table 1. We refer to $h_i^l \; i = 1, ..., N_l$ as the activation of neuron i in layer l.

3 Our solution approach

In this section, we discuss our approach for synthesizing an OOD detector based on Gaussian models. In statistics, Gaussian models are used to model the behavior of data samples. We adapt this idea to model the behavior of a neuron by a Gaussian model.

Consider a DNN as a classifier that distinguishes between $\{c_1, ..., c_{N_L}\} = C$ classes. One layer l of this DNN contains N_l neurons. For each class $c_o \in C$ in the training data set, we feed samples $x_j^{c_o}, \;\; j = 1, ..., m, \;\;$ into the network, and record the activations $n_i^{c_o}(x_j)$ of each neuron for $i = 1, ..., N_l$.

We collect those vectors $\vec{n}_i^{c_o}$, and calculate the mean and standard deviation, $\mu_i^{c_o}, \sigma_i^{c_o}$ of these values for each monitored neuron. We assume that the distribution of these values is approximately Gaussian. Thus, we expect the majority of samples to fall within

the range $[\mu_i^{c_o} - k\sigma_i^{c_o}, \mu_i^{c_o} + k\sigma_i^{c_o}]$, where k is typically a value close to 2, containing 95% of the samples. During testing, we feed a new sample x to the DNN. We then do the following: we record the class c that our DNN predicts on x, and also retrieve the neuron activation values $n_i^c(x)$. We check if the activations of each neuron i falls within its range for the predicted class.

More formally, we check if

$$\forall i = 1, ..., N_l \ : \ n_i^c(x) \in [\mu_i^c - k\sigma_i^c, \mu_i^c + k\sigma_i^c] \tag{1}$$

For a better understanding, we have the intuition depicted in Figure 1. The data in the plot is random but shall give an idea of how the approach works. There are two neurons that output different values, which are depicted as black dots. On the one hand, they are shown in a 2D-plane, which is used for the abstraction of Henzinger et al.; on the other hand, they are shown projected onto one dimension next to the axes, for our approach. The approach of Henzinger et al. fits interval boxes to the values that the neurons can take. The interval boxes are drawn in blue. Our approach calculates intervals based on fitted Gaussians. The mean of the Gaussian is depicted as a red cross next to the neuron activations. The red line marks the interval that we consider as good for the neuron.

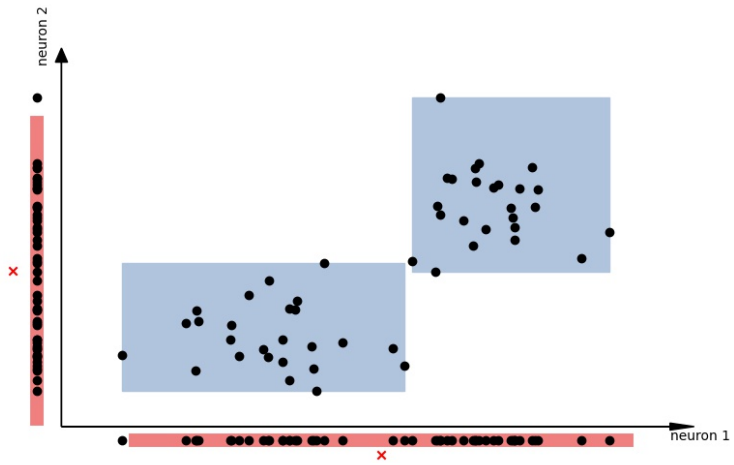


Fig. 1: This is an intuition of the Gaussian models on neuron activations. Black dots mark the values of the neurons. Once, in a 2D-plane together with the blue boxes that represent the abstraction of Henzinger et al., and once projected to one dimension only. The red lines mark the interval $[\mu_i^c - k\sigma_i^c, \mu_i^c + k\sigma_i^c]$ for the two neurons respectively. Those intervals are the basis for our approach of OOD-detection.

Each neuron "votes" independently if the new sample is valid or not. Samples within the distribution are expected to obtain a large number of votes, while OOD samples should obtain less. Thus, we collect the votes of all neurons, and then we compare them to a threshold; if they are below it, we consider x as an OOD sample, otherwise we consider it as correct. In that way, we can detect OOD inputs at runtime.

Note that this approach can also be extended to use multiple layers. For this, we compute the votes for each of the monitored layers. If they are below the threshold in at least one of the layers, we flag the sample as OOD.

An issue here is finding appropriate voting thresholds. For that, we use a suitable validation set. Normally, we should not make assumptions for the OOD data, and assume that we do not have access to them. In this case, we can use a suitable surrogate validation set, containing another unrelated dataset, e.g. adversarial examples or noisy images. In case we monitor more than one layer, the voting thresholds are computed individually for each layer.

4 Experiments

In this section, we analyze the experimental results of our approach. We will apply our approach for OOD detection to some example datasets and DNNs. We use the setting of Henzinger et al. [39], and we compare our result with theirs.

4.1 Datasets and Training

There are 4 datasets on which we evaluate our approach: MNIST, F-MNIST, CIFAR-10 and GTSRB (German Traffic Sign Recognition Benchmark) [41].

- MNIST is a dataset that contains images of digits. They shall be classified into ten classes, i.e. 0,...,9.
- F-MNIST consists of images of clothes, which shall also be classified into ten categories.
- CIFAR-10 is made of images of ten distinct classes from different settings.
- GTSRB contains images of German traffic signs that can be categorized into 43 classes.

All of the four datasets are used for classification. We train two different architectures of DNNs, NN1 and NN2, with the architectures of [39]. Those are:

- **NN1:** Conv(40), Max Pool, Conv(20), Max Pool, FC(320), FC(160), FC(80), FC(40), FC(k)
- **NN2:** BN(Conv(40)), Max Pool, BN(Conv(20)), Max Pool, FC(240), FC(84), FC(k)

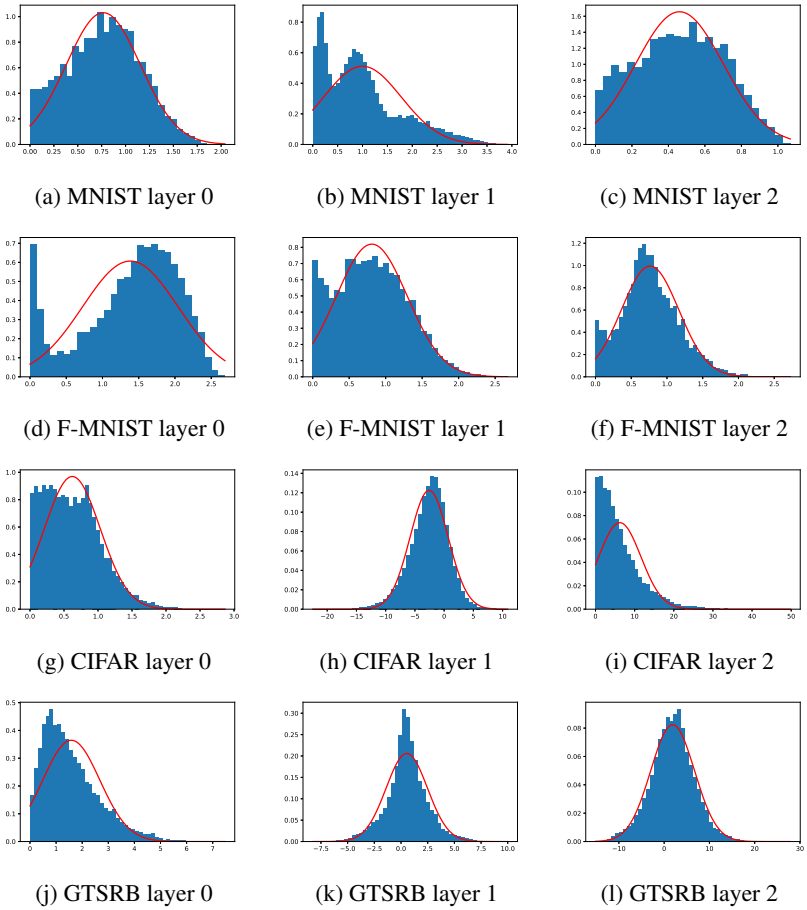
Here, FC is a fully connected layer, $Conv$ is a convolutional layer, $MaxPool$ is 2×2 max pooling, and BN is batch normalization. The activation function is always the RELU. NN1 was trained 10 epochs for MNIST, and 30 for F-MNIST, while NN2 was trained 200 epochs for CIFAR-10 and 10 for GTSRB. A learning rate of 10^{-3} and batch size 100 were used during training. NN1 is used for MNIST and F-MNIST, while NN2 is used for CIFAR-10 and GTSRB.

The evaluation is performed on two measures: the detection rate (DTERR) and the false alarm rate (FAR). The detection rate counts how many samples were correctly marked as OOD out of all OOD inputs. The false alarm rate (also known as Type-1-error) counts how many samples were marked as OOD but are not OOD, out of all marked inputs.

4.2 Gaussian Assumption

In this work, we used Gaussian distributions in order to approximate the output of each neuron. To verify that this is a valid assumption, we show in the following the distribution of values of the neurons.

We pick each dataset and select one random neuron from one of the monitored layers. Then, we plot the histogram of that neuron's output. We also show the Gaussian distribution we would expect to have, according to the measured μ and σ.



(a) MNIST layer 0 (b) MNIST layer 1 (c) MNIST layer 2

(d) F-MNIST layer 0 (e) F-MNIST layer 1 (f) F-MNIST layer 2

(g) CIFAR layer 0 (h) CIFAR layer 1 (i) CIFAR layer 2

(j) GTSRB layer 0 (k) GTSRB layer 1 (l) GTSRB layer 2

Fig. 2: Histogram of neuron outputs, along with the Gaussian distribution with the sample mean and variance.

We see that there are some small differences. For some neurons, the Gaussian assumption is very accurate, e.g. f,h,k, and l. For some other cases the histograms indicate a slightly different behavior, e.g. a,d,e,g,i. However, in general they show that the

neuron's outputs follow more a Gaussian behavior than a uniformly distributed one. It seems especially that the problem is rather that the parameters μ and σ do not exactly fit the true underlying Gaussian. One could think of calculating the parameters differently, or even using other models in future. Overall, the assumption that the neuron's outputs are Gaussian-like seems to be true.

4.3 Evaluation Steps

Following the setting of Henzinger et al., we perform the following steps for each dataset: we train the DNN for the first k classes of the dataset, and consider the rest as OOD. This results, for example, in a DNN that was only trained on the digits from 0 to 5. Digits from 6 to 9 are considered as OOD. Having now constructed the networks and datasets in this way, we can apply our approach, and compare the results with the ones of Henzinger et al.. We monitor all linear hidden layers of the DNNs for both approaches. We use the interval $[\mu - 2\sigma, \mu + 2\sigma]$ for each neuron and class label, while for Henzinger's approach, we use the parameters mentioned in their paper. Note that the monitor of Henzinger et al. outputs boolean values (e.g. x is inside or outside of the boxes), while ours outputs numerical scores (e.g. number of "votes" for an input x). In order to be able to compare the two approaches, we have to select a threshold for our approach, in order to convert its output to a boolean value (e.g. $votes(x) < \tau \Rightarrow OOD$).

For this, we set the threshold at a quantile of the in-distribution data, so that the FAR is similar to the one of [39]. For example, for a quantile $q = 50\%$, we set the voting threshold in a way that 50% of the known in-distribution data pass through. Having set the FARs on a similar level, we can then compare the detected errors of the approaches.

In the case where we monitor more than one layer, we use the same quantile q in every one of them, and then combine votes as described before, i.e. x is accepted if the votes of each layer are above the corresponding threshold. Having a different quantile threshold for every layer improves performance, but might also be prone to overfitting. Note also that the threshold q is not the same across experiments: in each run, we modify it in order to match the FAR of [39] on that particular experiment. The results are shown in Figure 3.

Each of the datasets has its own plot, where we have in red the values that the approach of [39] achieves, and in blue the values of our approach. For both of them, we measure the detection rate (DTERR) shown as a solid line in the left plot, and the false alarm rate (FAR) shown as a dashed line in the right plot. We see that the performance of our approach is mostly comparable or better than [39]. Especially, on CIFAR, our approach clearly outperforms the approach of Henzinger et al. in terms of the detection rate.

Overall, our approach seems promising and shows already good results. However, we also have to indicate some problems with both approaches, namely the occasional low detection rate or high false alarm rates. This is problematic for industrial applications, and shows us the difficulties involved, and the need for stronger approaches.

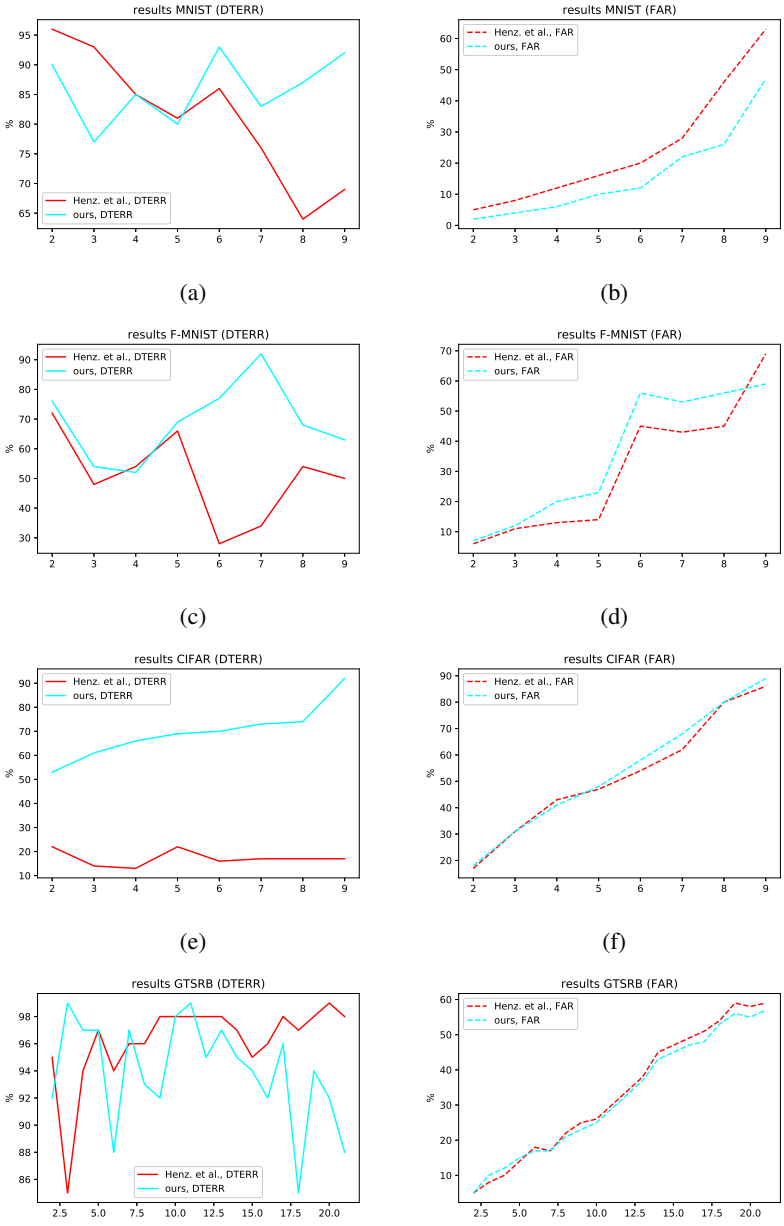


Fig. 3: Comparison between the approaches of Henzinger et al. and ours, for the cases of MNIST, F-MNIST, CIFAR , and GTSRB datasets. Here, $DTERR$ and FAR are shown in separate diagrams. The number of classes on which the network was trained is depicted on the x axis.

4.4 Parameter Study

In this section, we perform a study on the parameters of our approach. For simplicity, we focused on the MNIST dataset. The DNN in this case was thus NN1 with a total of eight layers. We want to particularly investigate the effects of the number of layers.

Layers	$DTERR$ (%)	FAR (%)
5	44.6	8.7
6	69.2	5.3
7	73.0	5.4
8	73.5	6.4

Layers	$DTERR$ (%)	FAR (%)
(5,6)	70.3	8.3
(6,7)	77.0	6.5
(7,8)	76.4	6.8
(6,7,8)	79.5	7.5
(5,6,7,8)	80.0	9.8

Table 1: Results on different layers, and different combination of layers. The evaluation is performed on the detection rate and the false alarm rate. Layers closer to the output layer show a higher detection rate than layers earlier in the DNN. The combination of several layers only results in a small improvement compared to the usage of only one layer.

At first, we look at different layers in the DNN. The fifth layer seems to contain less important information in comparison to layer six, seven, and eight. When we only monitor layer five, the DTERR is almost 20% lower as for the other layers, while the FAR does not change significantly. We can thus verify the intuition that the features of the later layers in a DNN are more meaningful. If we combine the voting of several layers, we can see that the detection rate is slightly increased. Especially, the bad DTERR of 44% by only using layer five can be drastically improved by adding the knowledge of layer six, namely to 70%; while the FAR even decreases slightly. The combination of other layers can still increase the DTERR up to 80.0%, however, it comes with a slightly higher false alarm rate. Thus, for a more light-weight approach, it could be recommended to stick with fewer layers. Additionally, there may also be another different voting system for several layers, e.g. incorporate a weighted voting system for the layers and granting later layers more influence on the result.

5 Outlook

A natural next step would be to use additional information given by the correlation between the neurons. So far, we only considered the Gaussians of each neuron independently.

Instead, we can consider a subset of neurons $n_k \in S$ and fit a joint Gaussian distribution $N(\mu_S, \Sigma_S)$ on them. This subset can be an entire layer, where we fit a Gaussian distribution on the entire vector of layer activations, but it can also be a smaller subset of neurons. This offers the advantage of reduced computations, and an easier estimation of the covariance matrix (which is hard in high dimensions). The approach is flexible, and allows us to consider arbitrary subsets of neurons with varying sizes. Predictions

can then be combined again by voting. For multidimensional Gaussian distributions, a simple threshold with μ and σ is no longer possible. Instead, one can use the Mahanalobis distance, $M^2(x) = (x - \mu)^T \Sigma^{-1}(x - \mu)$, which is a notion of distance from the distribution center. A suitable threshold for $M(x)$ is then to be calculated.

Besides, for a subset of neurons, a more precise model that can be used is a mixture of Gaussians. This might be more accurate since the Gaussian distributions as above are only imprecise approximations of the true distribution, while in contrast, Gaussian mixture models can approximate any probability distribution to any precision.

6 Conclusion

In this work, we considered the problem of runtime monitoring of DNNs, which forms an important step towards applying deep learning to safety-critical systems. Specifically, we focused on the sub-problem of OOD detection, and developed a lightweight detection method based on Gaussian models of neuron activation values. This can be extended in various ways as described before, and gives more accurate results than the recent work of Henzinger et al. [39]. Interestingly, the results suggest that reflecting correlation of the activation values (as in [39]) is less important than handling outliers through voting on learnt models (as here). Actually, the rigid and complete coverage by the boxes does not seem as adequate as the learnt approximations.

While we showed already a good efficiency on OOD inputs, the industrial requirements suggest that further improvements are necessary to reach real-world applicability. Our preliminary results invite further investigation along these directions. In particular, runtime monitoring by more complex probabilistic models, such as Gaussian mixtures, or using DNN-based probability estimation methods such as Normalizing Flows seem very promising.

References

1. C.-H. Cheng, G. Nührenberg, H. Yasuoka: Runtime Monitoring Neuron Activation Patterns. DATE, 2019. URL: https://arxiv.org/abs/1809.06573
2. I. Goodfellow, Y. Bengio, A. Courville: Deep Learning. MIT Press, 2016. URL: https://www.deeplearningbook.org/
3. X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, X. Yi: A Survey of Safety and Trustworthiness of Deep Neural Networks. CoRR, 2018. URL: https://arxiv.org/abs/1812.08342
4. P. Ortega, V. Maini: Building safe artificial intelligence: specification, robustness, and assurance. Deep Mind blog, 2018. URL: https://medium.com/@deepmindsafetyresearch/building-safe-artificial-intelligence-52f5f75058f1
5. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus: Intriguing properties of neural networks. ICLR, 2014. URL: https://arxiv.org/abs/1312.6199
6. Wikipedia: List of self-driving car fatalities. Wikipedia article, 2018. URL: https://en.wikipedia.org/wiki/List-of-self-driving-car-fatalities
7. A. Kurakin, I. Goodfellow, S. Bengio: Adversarial Machine Learning at Scale. ICLR, 2017. URL: https://arxiv.org/abs/1611.01236

8. I. J. Goodfellow, J. Shlens, C. Szegedy: Explaining and Harnessing Adversarial Examples. ICLR, 2015. URL: https://arxiv.org/abs/1412.6572
9. N. Carlini, D. Wagner: Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. AIsec, 2017. URL: https://arxiv.org/abs/1705.07263
10. G. Katz, C. Barrett, D. Dill, K. Julian, M. Kochenderfer: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. CAV, 2017. URL: https://arxiv.org/abs/1702.01135
11. V. Tjeng, K. Xiao, R. Tedrake: Evaluating Robustness of Neural Networks with Mixed – Integer Programming. ICLR, 2019. URL: https://arxiv.org/abs/1711.07356
12. E. Wong, Z. Kolter: Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. ICML, 2018. URL: https://arxiv.org/abs/1711.00851
13. S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, P. Kohli: On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. NIPS, 2018. URL: https://arxiv.org/abs/1810.12715
14. M. Mirman, T. Gehr, M. Vechev: Differentiable Abstract Interpretation for Provably Robust Neural Networks. ICML, 2018. URL: https://files.sri.inf.ethz.ch/website/papers/icml18-diffai.pdf
15. R. McAllister, Y. Gal, A. Kendall, M. van der Wilk, A. Shah, R. Cipolla, A. Weller: Concrete Problems for Autonomous Vehicle Safety: Advantages of Bayesian Deep Learning. IJCAI, 2017. URL: https://www.ijcai.org/Proceedings/2017/661
16. C. Olah et. al.: The Building Blocks of Interpretability. Distill article, 2018. URL: https://distill.pub/2018/building-blocks/
17. C. Guo, M. Rana, M. Cisse, L. van der Maaten: Countering Adversarial Images using Input Transformations. ICLR, 2018. URL: https://arxiv.org/abs/1711.00117
18. J. H. Metzen, T. Genewein, V. Fischer, B. Bischoff: On Detecting Adversarial Perturbations. ICLR, 2017. URL: https://arxiv.org/abs/1702.04267
19. R. Feinman, R. R. Curtin, S. Shintre, A. B. Gardner: Detecting Adversarial Samples from Artifacts. Arxiv preprint, 2017. URL: https://arxiv.org/abs/1703.00410
20. Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, Dan Mané: Concrete Problems in AI Safety. CoRR, 2016. URL: https://arxiv.org/abs/1606.06565
21. D. Hendrycks, K. Gimpel: A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. ICLR, 2017. URL: https://arxiv.org/abs/1610.02136
22. B. Lakshminarayanan, A. Pritzel, C. Blundell: Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. NIPS, 2017. URL: https://arxiv.org/abs/1612.01474
23. S. Liang, Y. Li, R. Srikant: Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks. ICLR, 2018. URL: https://arxiv.org/abs/1706.02690
24. K. Lee, K. Lee, H. Lee, J. Shin: A Simple Unified Framework for Detecting Out-of- Distribution Samples and Adversarial Attacks. NIPS, 2018. URL: https://arxiv.org/abs/1807.03888
25. J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. A. DePristo, J. V. Dillon, B. Lakshminarayanan: Likelihood Ratios for Out-of-Distribution Detection. NeurIPS, 2019. URL: https://arxiv.org/abs/1906.02845
26. S. Ren, K. He, R. Girshick, J. Sun: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015. URL: https://arxiv.org/abs/1506.01497
27. J. Redmon, S. Divvala, R. Girshick, A. Farhadi: You Only Look Once: Unified, Real-Time Object Detection. CVPR, 2016. URL: https://arxiv.org/abs/1506.02640

28. X. Chen, H. Ma, J. Wan, B. Li, T. Xia: Multi-View 3D Object Detection Network for Autonomous Driving. CVPR, 2017. URL: https://arxiv.org/abs/1611.07759
29. J. Ku, M. Mozifian, J. Lee, A. Harakeh, S. Waslander: Joint 3D Proposal Generation and Object Detection from View Aggregation. IROS, 2018. URL: https://arxiv.org/abs/1712.02294
30. C. R. Qi, W. Liu, C. Wu, H. Su, L. J. Guibas: Frustum PointNets for 3D Object Detection from RGB-D Data. CVPR, 2018. URL: https://arxiv.org/abs/1711.08488
31. Y. LeCun, C. Cortes: The MNIST database of handwritten digits. NIST, 1998. URL: http://yann.lecun.com/exdb/mnist/
32. J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel: The German Traffic Sign Recognition Benchmark: A multi-IJCNN, 2011. URL: http://benchmark.ini.rub.de/
33. A. Geiger, P. Lenz, R. Urtasun: Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. CVPR, 2012. URL: http://www.cvlibs.net/datasets/kitti/
34. I. J. Goodfellow, J. Shlens, C. Szegedy: Explaining and Harnessing Adversarial Examples. ICLR, 2015. URL: https://arxiv.org/abs/1412.6572
35. A. Kurakin, I. Goodfellow, S. Bengio: Adversarial examples in the physical world. ICLR, 2017. URL: https://arxiv.org/abs/1412.6572
36. A. Krizhevsky: Learning Multiple Layers of Features from Tiny Images. Toronto University, 2009. URL: https://www.cs.toronto.edu/~kriz/cifar.html
37. AUDI AG (Private Communication). 2020.
38. K. P. Murphy: Machine Learning: a Probabilistic Perspective. MIT Press, 2012.
39. T. A. Henzinger, A. Lukina, C. Schilling: Outside the Box: Abstraction-Based Monitoring of Neural Networks. ECAI 2020, URL: https://arxiv.org/abs/1911.09032
40. C. M. Bishop: Pattern recognition and machine learning Springer, 2006
41. J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In Proceedings of the IEEE International Joint Conference on Neural Networks, pages 1453–1460. 2011.
42. ISO/PAS 21448. "Road vehicles - Safety of the intended functionality". URL: "https://www.iso.org/obp/ui/#iso:std:70939:en"
43. Marco A. F. Pimentel, David A. Clifton, Lei A. Clifton, and Lionel Tarassenko: 'A review of novelty detection', Signal Processing, 99, 215–249, (2014).