Neurocomputing 470 (2022) 335–343

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Efficient detection of adversarial, out-of-distribution and other misclassified samples

Julia Lust *, Alexandru P. Condurache

University of Lübeck, Institute for Signal Processing, Germany
Robert Bosch GmbH, Automated Driving, Germany

ARTICLE INFO

Article history:
Received 7 January 2021
Revised 9 April 2021
Accepted 3 May 2021
Available online 22 July 2021

Keywords:
Deep Neural Networks
Out-of-distribution detection
Adversarial example detection
Uncertainty quantification
Safety

ABSTRACT

Deep Neural Networks (DNNs) are increasingly being considered for safety–critical approaches in which it is crucial to detect misclassified samples. Typically, detection methods are geared towards either the detection of out-of-distribution or adversarial data. Additionally, most detection methods require a significant amount of parameters and runtime. In this contribution we discuss a novel approach for detecting misclassified samples suitable for out-of-distribution, adversarial and additionally real world error-causing corruptions. It is based on the Gradient's Norm (GraN) of the DNN and is parameter and runtime efficient. We evaluate GraN on two different classification DNNs (DenseNet, ResNet) trained on different datasets (CIFAR-10, CIFAR-100, SVHN). In addition to the detection of different adversarial example types (FGSM, BIM, Deepfool, CWL2) and out-of-distribution data (TinyImageNet, LSUN, CIFAR-10, SVHN) we evaluate GraN for novel corruption set-ups (Gaussian, Shot and Impulse noise). Our experiments show that GraN performs comparable to state-of-the-art methods for adversarial and out-of-distribution detection and is superior for real world corruptions while being parameter and runtime efficient.

1. Introduction

Deep Neural Networks (DNNs) achieve outstanding results in a wide variety of areas such as speech recognition or object detection and especially in perceptual tasks, they have gained a great advantage over classical methods in recent years [1–3]. Due to this progress, DNNs are increasingly being considered for safety relevant tasks, such as autonomous driving [4] or medical prognoses [5].

An input is *misclassified* by a DNN if it is not within its *generalization area*, the area in which the information contained in the input sample is processed reasonably by the DNN [6]. There are different reasons for a sample to be outside the generalization area. Unforeseen data-shift or corner cases can occur, such that the data is not covered sufficiently by the training data. Such data is summarized by the term *out-of-distribution* data. In research such a scenario is typically simulated by using data from a dataset different to the training data. Another reason for misclassifications are *adversarial examples* that are constructed from methods exploiting the generalization issues of DNNs on purpose [7–10]. Those methods, called adversarial attacks, typically generate adversarial samples from originally correctly classified samples by perturbing them slightly. Often, the perturbation is invisible to humans but the DNN is fooled and a misclassification occurs. A third category that leads to errors are real world corruptions such as sensor noise. These kind of corruptions depend on different circumstances such as lighting or bit errors and can cause misclassifications. In comparison to out-of-distribution samples and adversarial examples such causes for misclassifications are more common in real world datasets [11].

Especially for safety relevant approaches it is crucial to detect misclassified data samples during inference time. For each, the detection of out-of-distribution samples [12–20] and the detection of adversarial examples [21–25,13], there exist a variety of methods. Each method is typically geared towards either the detection of adversarial or out-of-distribution samples, and unfortunately, in both fields the current state-of-the-art detectors typically require significantly more parameters or runtime than the original DNN itself. This is critical for applications such as autonomous driving where the available resources are limited [26]. Furthermore, the detection of real world error corruptions is often not considered. There are efficient methods that detect corrupted data in general [27], but they do not differentiate between corruption that still allows a correct classification and corruption leading to a misclassification. For all set-ups we want to show that state-of-the-art detection performance is possible while being runtime and param-

* Corresponding author.
E-mail address: juliarebecca.lust@de.bosch.com (J. Lust).

https://doi.org/10.1016/j.neucom.2021.05.102
0925-2312/© 2021 Published by Elsevier B.V.

eter efficient. **GraN**, a method based on the **Gra**dient's **N**orm of the DNN [28], builds our basis for further investigation on a runtime and parameter efficient detection of misclassifications.

Our main contributions are:

- Application of GraN on out-of-distribution, adversarial example detection and novel real world error detection.
- Demonstrating the versatility of GraN by evaluating it on the widely used image classification DNNs DenseNet and ResNet on several datasets.
- Showing state-of-the-art performance of GraN on numerous set-ups while being the most runtime and parameter efficient detector.
- Investigations on the relevance of the pre-processing procedure used in GraN which includes the option for further runtime reduction.
- Research on which layers are most important for the performance of GraN which also includes the option for further runtime reduction.

2. Related work

Adversarial example and out-of-distribution detection are usually treated independently from each other despite of their common origin of detecting misclassified samples. Hence, methods are only geared towards one of these set-ups. Nevertheless, the methods of both fields can be split into the same four main categories [25,6]: **generative, inconsistency, ensemble** and **metric** based approaches.

Generative methods are mainly based on a pre-processing procedure in which the image is shifted back into the direction of the distribution of the training data by an encoder-decoder procedure. In the case of adversarial example detection the goal is to remove the adversarial noise [22,23]. For out-of-distribution detection, a huge difference between the original and the output image of the generative process is expected to hint towards an out-of-distribution sample [16]. However, the encoder-decoder approach induces many parameters. Therefore generative methods are not compatible for parameter and runtime restricted applications.

Inconsistency based approaches expect the output of the DNN for a misclassified image to be more sensitive to small changes in the image. The difference to the generative methods is that the changes are rather simple in their computation. A well-known example is the method *ODIN* [12]. In its approach the original image x is shifted in the direction of the steepest descend of the loss function L computed for the output of the network $F(x)$ and the predicted class y by performing a one step gradient descent with step size ϵ

$$\tilde{x} = x - \epsilon \cdot \operatorname{sign}(\nabla_x L(x, y)). \tag{1}$$

ODIN shows good detection performance for out-of-distribution samples while being runtime and especially parameter efficient. Recently, an updated version of ODIN was introduced which outperforms the original version [14]. The runtime, however, was worsened by a pre-processing procedure that iterates over every possible class. The updated version of ODIN is therefore not relevant for runtime restricted problems. For the detection of adversarial examples, unfortunately, it was shown that existing inconsistency methods do not work well for some adversarial attacks [24,25].

Ensemble based detectors are mainly used for out-of-distribution detection. The procedure is based on an ensemble of networks each typically trained slightly differently [17,20]. The more the outputs of DNNs differ for the same image, the more the image is expected to be out-of-distribution. The number of parameters for several DNNs exceeds the limit of parameter restricted applications which makes them irrelevant for such use cases.

Metric based methods compare if the current input sample is behaving similarly to correctly classified input samples investigated during training. The features for such methods are commonly the output or more rarely the gradient of each layer of the DNN. During inference time the features are then compared to those of training samples using a stochastic method. Metric methods are the most promising in the field of efficiency and hence especially rather simple efficient methods in this category are recently getting more attention [27,29]. However such methods are usually less perfomand or are evaluated on rather restricted set-ups. One well-known adversarial example detection approach is LID delivering comparable to state-of-the-art results [21]. It is based on a Local Intrinsic Dimensionality score, a weighted distance based on k-nearest neighbors from the training set. However, LID was recently outperformed in the category of most efficient methods for the detection of adversarial samples by GraN [28]. GraN is, similar to one older detection method [15], based on gradient information. The main differences to the older method are that GraN is applying an efficient pre-processing step and uses only layer-wise L1-norm based features, which lowers the computational overhead. A detailed explanation of the method GraN can be found in Section 3.1. Other, new procedures for adversarial detection are typically more computational and parameter expensive by using additional procedures such as classification networks consisting of one fully connected softmax layer on top of each activation layer by delivering a comparable perfomance [25].

The current state-of-the-art of metric based methods in out-of-distribution detection are dominated by computational expensive methods based on computation of layer-wise higher-order Gram matrices [18] or on an additional residual flow procedure [19]. Due to large runtime or many parameters, such methods can not be considered in complexity restricted fields.

The only method applied on both, adversarial and out-of-distribution detection is based on the Mahalanobis distance $M(x)_l$ computed for each layer l of the DNN [13]. It is defined by the layer-output $f(x)_l$ of the test sample x and the closest class-conditional Gaussian distribution determined by the layer-output mean μ_y and the layer-output covariance Σ_y for samples of the class y

$$M(x)_l = -(f(x)_l - \mu_y)\Sigma_y^{-1}(f(x)_l - \mu_y).$$

It is performed in combination with a pre-processing procedure based on the gradient for the predicted class.

Outlook: To show the efficiency of **GraN** [28], it is compared to most known comparable methods: **LID** [21], an efficient method in case of adversarial detection and **ODIN** [12] the most efficient method in case of out-of-distribution detection. Furthermore to demonstrate performance on par with the state-of-the-art, we compare GraN to **Mahalanobis** [13], the only other method that has been applied to both adversarial and out-of-distribution detection and often used as a baseline method in both fields.

3. GraN: A gradient-norm based detector

This section explains the systematic approach of GraN and its intuition.

3.1. Method

GraN predicts if a pre-trained DNN with weights θ is misclassifying an input x by outputting a value $p \in [0, 1]$. A high value p indicates a high possibility for a misclassification and vice versa. The approach can be divided into a pre-processing procedure, a feature extraction and a feature processing step. The feature extraction step can be seen as the core of the method. A visualization of the method is shown in Fig. 1.

3.1.1. Pre-processing

The method starts with a pre-processing step. The original input image x is smoothened using a two dimensional symmetric kernel derived from a Gaussian distribution $G(u, v)$ for each color channel. A two dimensional symmetric Gaussian is defined by

$$G(u, v) = \frac{1}{\sigma 2\pi} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

with u, v the horizontal and vertical distance from the center and σ the desired standard deviation. To get a discrete kernel that can be applied to pixels a discrete function approximates $G(u, v)$ [30]. The resulting discrete values can then be used as a filter mask and applied time-efficiently as an additional convolution step to the image x to generate the smoothed image $\tilde{x}$.

3.1.2. Feature extraction

Both images, the original image x, and the smoothed image $\tilde{x}$ are processed by the pre-trained DNN. As usual the predicted class y is derived as the index of the largest value of the output $F(x)$ of the network for the original image x. The output $F(\tilde{x})$ of the smoothed image $\tilde{x}$ and the predicted class y are fed into a loss function $L(\Theta, F(\tilde{x}), y)$. In the next step the gradient with respect to the weights $\nabla_\Theta L(\Theta, F(\tilde{x}), y)$ is computed time-efficiently via backpropagation[1]. The large set of gradients $\nabla_\Theta L(\Theta, F(\tilde{x}), y)$ is transformed to a smaller set: For each layer $i \in \{1, \dots, n\}$ of the DNN, the gradient regarding the layer's weights Θ_i is replaced by its L_1 norm:

$$\nabla_\Theta L(\Theta, F(\tilde{x}), y) = \begin{pmatrix} \nabla_{\theta_1} L(\Theta, F(\tilde{x}), y) \\ \vdots \\ \nabla_{\theta_n} L(\Theta, F(\tilde{x}), y) \end{pmatrix} \mapsto \begin{pmatrix} ||\nabla_{\theta_1} L(\Theta, F(\tilde{x}), y)||_1 \\ \vdots \\ ||\nabla_{\theta_n} L(\Theta, F(\tilde{x}), y)||_1 \end{pmatrix}. \tag{2}$$

This results in a feature vector of size n which reflects the number of layers in the DNN.

3.1.3. Feature processing

The n-dimensional feature vector is processed by a logistic regression network with $n+1$ parameters. During the training phase of GraN this logistic regression network has to be trained with correctly and incorrectly classified samples such that it predicts for each input the value p stating if the input is misclassified ($p \gg 0$) or not ($p \ll 1$).

3.2. Intuition

The core of GraN is the norm of the gradient of the loss function

$\nabla_\Theta L(\Theta, F(\tilde{x}), y)$.

From a mathematical point of view, the gradient over a variable indicates the effect of small changes of that variable on the output. Therefore, we can conclude that a high uncertainty at an evaluation point of a function is indicated by a high gradient, which is directly enlarging the values of the feature vector of GraN (Eq. (2)).

One natural indicator of uncertainty is a high difference between the output of the DNN and the resulting predicted one-hot class vector y. This information is included in the gradient over the last layer which builds the starting point for the backpropagation. Therefore, a huge distance between $F(x)$ and the one-hot vector for class y directly increases all gradient values.

In order to increase the gradient for misclassified examples, a smoothed input $\tilde{x}$ is used. The Gaussian smoothing step is thought to work as a low-pass-filter, small details and noise are removed and the network is hence forced to concentrate on larger-scale features. These larger scale features are harder to be influenced by adversarial noise. Furthermore, if the prediction of the original and the smoothed image do not resemble each other, we know that the larger scale features and the smaller scale features contradict each other. Such a contradiction eventually results in large gradients in the feature vector (Eq. (2)).

4. Evaluation

We evaluate GraN on a DenseNet with 100 layers [31] and a ResNet with 30 layers [32] for the classification task and CIFAR-10, CIFAR-100 [33] and SVHN [34] as in-distribution datasets. There are predefined training and test samples for all datasets. Each network is trained on each training dataset and tested on the corresponding test set. The DNNs and their training procedure are adopted from reference [13].

4.1. Adversarial example detection

In the adversarial set-up we want to evaluate the performance of GraN in distinguishing between correctly classified images from the test data and adversarial images. In the following we refer to the correctly classified images as *in-distribution* images. In order to generate adversarial examples four different attack methods are used: FGSM [7], BIM [8], Deepfool [9] and CW-L2 [10]. Each adversarial attack method is applied to all in-distribution images and defines a data-setup for the detection task together with the in-distribution images. Each data set-up is split randomly into a train (80%), validation (10%) and test (10%) set by keeping a ratio of one between adversarial and in-distribution images in each subset.

For each set-up the logistic regression for GraN and the trainable parts of the methods LID [21] and Mahalanobis [13] described in Section 2 are trained on the training dataset. The validation dataset is used to find the best hyper-parameter setting for each set-up and each method. The range of possible hyper-parameters are adopted for Mahalanobis and LID from reference [13]. The only hyper-parameter of GraN is the standard deviation σ for the Gaussian smoothing pre-processing. It is chosen from the range $\sigma \in \{0.1, 0.2., \dots, 2.0\}$ by using the validation data.

We use the area under the receiver operating characteristic curve (AUROC) as most widely adopted evaluation metric in the adversarial example detection literature. The receiver operating characteristic curve plots the true positive rate (TPR) of in-distribution data against the false positive rate (FPR) of the adversarial data.

The adversarial example detection AUROC score and the runtime and parameter overhead for each method are provided in Table 1.

4.2. Out-of-distribution detection

In the out-of-distribution task the detector has to distinguish between in-distribution and out-of-distribution images. As in-distribution datasets we use CIFAR-10, CIFAR-100 [33] and SVHN [34]. As out-of-distribution datasets we use CIFAR-10 [33], SVHN [34], Tiny ImageNet [35] and LSUN [36], each defining an out-of-distribution set-up together with the in-distribution data. As in reference [21] the out-of-distribution images are resized in order to

[1] The last two steps are similar to a training step with the smoothed image as input but instead of the label the predicted class y is used and the DNN is not updated.

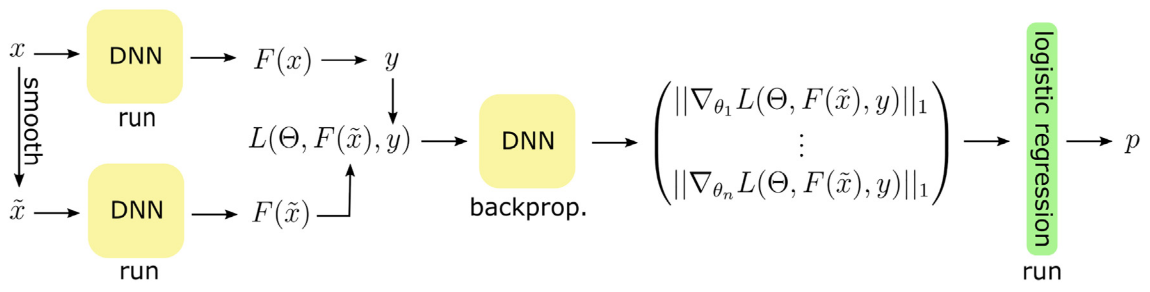

x
smooth
$\tilde{x}$
DNN
run
DNN
run
$F(x)$
y
$L(\Theta, F(\tilde{x}), y)$
$F(\tilde{x})$
DNN
backprop.
$\begin{pmatrix} \|\nabla_{\theta_1} L(\Theta, F(\tilde{x}), y)\|_1 \\ \vdots \\ \|\nabla_{\theta_n} L(\Theta, F(\tilde{x}), y)\|_1 \end{pmatrix}$
logistic regression
run
p

Fig. 1. Overview of GraN [28].

Table 1
AUROC score and efficiency comparison for adversarial example detection. We thrive for an parameter and runtime optimized solution at the same time, hence we marked the results of the most efficient method GraN that are in the range of (less than 2 percentage points worse) the best result in **bold**. The best results are marked by an $\underline{\text{underline}}$.

Net	Data	Advers.	LID [21]			Mahalanobis [13]			GraN [28]		
		Type	Para.	Time	AUROC	Para.	Time	AUROC	Para.	Time	AUROC
DenseNet	CIFAR-10	FGSM	62.4k	0.02[s]	98.50	157.9k	0.07[s]	$\underline{99.99}$	0.3k	0.04[s]	**99.52**
		BIM			99.75			$\underline{99.73}$			**98.50**
		Deepfool			85.86			73.49			**$\underline{86.21}$**
		CWL2			81.17			79.68			**$\underline{86.58}$**
	CIFAR-100	FGSM	62.4k	0.02[s]	$\underline{99.79}$	214.1k	0.07[s]	99.74	0.3k	0.04[s]	**99.54**
		BIM			$\underline{97.93}$			97.77			**96.80**
		Deepfool			75.44			78.74			**$\underline{85.63}$**
		CWL2			73.68			80.68			**$\underline{85.66}$**
	SVHN	FGSM	62.4k	0.02[s]	98.14	157.9k	0.07[s]	$\underline{99.92}$	0.3k	0.04[s]	**98.72**
		BIM			96.28			$\underline{99.39}$			**98.00**
		Deepfool			93.87			96.44			**$\underline{96.59}$**
		CWL2			95.05			$\underline{96.97}$			**96.89**
ResNet	CIFAR-10	FGSM	102.4k	0.02[s]	99.85	362.5k	0.02[s]	$\underline{99.99}$	0.1k	0.01[s]	**99.07**
		BIM			97.03			$\underline{99.68}$			**98.11**
		Deepfool			56.78			$\underline{92.38}$			**90.42**
		CWL2			82.89			$\underline{95.74}$			80.06
	CIFAR-100	FGSM	102.4k	0.02[s]	99.13	454.7k	0.02[s]	$\underline{99.83}$	0.1k	0.01[s]	**99.58**
		BIM			97.01			97.34			**$\underline{98.01}$**
		Deepfool			73.88			86.01			**$\underline{89.13}$**
		CWL2			80.59			$\underline{92.45}$			**91.72**
	SVHN	FGSM	102.4k	0.02[s]	97.95	362.5k	0.02[s]	$\underline{99.69}$	0.1k	0.01[s]	**98.28**
		BIM			90.72			$\underline{96.97}$			94.82
		Deepfool			92.33			95.36			**$\underline{97.01}$**
		CWL2			88.15			$\underline{92.99}$			**92.77**
	Average				89.66			93.79			**$\underline{94.07}$**

have the same size as the corresponding in-distribution images. Similar as in the adversarial example task a training, evaluation and test set is built for each set-up. For each out-of-distribution set-up the methods GraN, ODIN and Mahalanobis are trained on the training set and the best hyperparametersetting are found using the evaluation set. The out-of-distribution detection AUROC performance and the runtime and parameter overhead for all methods are provided in Table 2.

4.3. Real world error detection

In real world applications the occurence of adversarial or out-of-distribution data far from the training data is often not the main cause for misclassifications. More relevant are misclassifications caused by data corruption or misclassifications of in-distribution samples. The following experiments cover such real world error causing scenarios. The *original* set-up is build by the misclassified images and the same amount of randomly chosen correctly classified images. In the real world error detection set-ups the detectors have to distinguish between corrupted images that are still correctly classified by the DNN and corrupted images in which the corruption leads to a misclassification. Sensor related corruption types are relevant in most computer vison tasks. We therefore based the corruption set-ups on the three sensor related corruption types introduced by Hendrycks and Dietterich: *Gaussian* noise which can appear in low-lighting conditions, *shot* noise which is also called Poisson noise and is electronic noise caused by the discrete nature of light itself and *impulse* noise which is a color analogue of salt-and-pepper noise and can be caused by bit errors [11]. For each dataset, network and corruption type we perturbed the original data using the corresponding noise and adapted the noise level such that half of the original data is misclassified and half of the data is still correctly classified by the DNN. In each set-up the detectors have to distinguish the correctly classified images from the misclassified ones. Again, as underlying datasets we use CIFAR-10, CIFAR-100 [33] and SVHN [34]. Similar to the adversarial example task a training, evaluation and test set is built for each set-up. The best performing methods in the earlier examples, GraN and Mahalanobis, are evaluated by being trained on the training set and the best hyperparameter settings are found using

Table 2
AUROC score and efficiency comparison for out-of-distribution detection. We thrive for a parameter and runtime optimized solution at the same time, hence we marked the results of the most efficient method GraN that are in the range of (less than 2 percentage points worse) the best result in **bold**. The best results are marked by an underline.

Net	In. Data	Out-of-distr.	ODIN [12]			Mahalanobis [13]			GraN [28]		
		Data	Para.	Time	AUROC	Para.	Time	AUROC	Para.	Time	AUROC
DenseNet	CIFAR-10	SVHN	2	0.06[s]	95.62	157.9k	0.07[s]	97.83	0.3k	0.04[s]	**98.04**
		T.Imagenet			98.51			98.24			**97.41**
		LSUN			99.23			99.30			**99.31**
	CIFAR-100	SVHN	2	0.06[s]	93.81	214.1k	0.07[s]	95.74	0.3k	0.04[s]	**98.79**
		T.Imagenet			85.22			92.57			**98.06**
		LSUN			84.45			97.17			**99.41**
	SVHN	CIFAR-10	2	0.06[s]	91.37	157.9k	0.07[s]	98.99	0.3k	0.04[s]	**97.90**
		T.Imagenet			95.11			99.88			**98.70**
		LSUN			94.54			99.91			**99.22**
ResNet	CIFAR-10	SVHN	2	0.02[s]	96.65	362.5k	0.02[s]	99.19	0.1k	0.01[s]	**98.83**
		T.Imagenet			94.04			99.43			**98.04**
		LSUN			94.14			99.73			**99.26**
	CIFAR-100	SVHN	2	0.02[s]	93.94	454.7k	0.02[s]	98.36	0.1k	0.01[s]	**98.78**
		T.Imagenet			87.62			98.06			**98.55**
		LSUN			85.64			98.13			**99.51**
	SVHN	CIFAR-10	2	0.02[s]	92.09	362.5k	0.02[s]	99.33	0.1k	0.01[s]	**98.34**
		T.Imagenet			91.99			99.88			**98.74**
		LSUN			89.43			99.89			**99.14**
	Average				92.41			98.42			**98.67**

the evaluation set. In addition, a baseline method is introduced to ensure that the comparatively better performance of GraN is not only due to the Gaussian pre-processing. This *pre-processing baseline* uses the Gaussian pre-processing step of GraN but performs the detection only based on the difference of the softmax value of the predicted class. The real world detection AUROC performance and the runtime and parameter overhead for all methods are provided in Table 3.

4.4. Discussion of the results

Runtime: In all tables (Tables 1–3), the column *Time* provides the runtime that is needed for each detection method. The time for the forward pass of the classification network to classify the input image is not included. Times are measured on a single Nvidia GeForce GTX 1080ti GPU and the DNNs are implemented in PyTorch Version 1.5.1. With a runtime overhead of 0.04 s for DenseNet and 0.01 s for ResNet, GraN has a shorter runtime than all other three methods on all set-ups except for LID on DenseNet. The pre-processing step in GraN can be performed time-efficiently as convolution and the resulting image can be processed by the DNN in parallel to the classification step of the original image. The main computation-time overhead of GraN comes from the necessary backpropagation step to calculate the gradients. The following layer-wise summation and the processing by the logistic regression account for only a small part. A backpropagation step for the 100 layer DenseNet needs longer than a backpropagation step for the 30 layer ResNet, which explains the runtime difference between these settings. The pre-processings of ODIN and Mahalanobis need gradient information depending on the predicted class. Therefore, their pre-processing steps are based on a forward and backward pass. Hence, the forward passes of the preprocessed and the original image can not be computed in parallel, which roughly leads to a doubling of time compared to the runtime of GraN. In addition, Mahalanobis is based on the layer-wise Mahalanobis distance, which is complex to calculate and therefore contributes to even more runtime. LID is not based on a pre-processing procedure. However, it needs an expensive computation of the layer-wise local intrinsic dimensionality score. The absence of a pre-processing step results in a runtime advantage over GraN for deep networks. For flatter networks, on the other hand, the more complex feature extraction process is relatively slower and GraN is faster. This behavior can be observed when comparing the runtime for GraN and LID for the flatter network ResNet and the deeper network DenseNet. For real-time relevant applications, even flatter networks are usually used, for which the runtime advantage of GraN extends.

Parameters: The column *Para.* provides the additional parameters each method needs in the corresponding set-up (Tables 1–3). LID and Mahalanobis do require a huge amount of parameter overhead in comparison to GraN and ODIN. This is due to their feature extraction method. On top to the parameters required for the logistic regression part, Mahalanobis needs to store the layer-wise outputs' average and standard deviation, and LID the layer-wise output of 100 different samples. ODIN does not need any parameters, while GraN only needs the parameters of the logistic regression part that are very few. Hence, from the parameter perspective ODIN and GraN are relevant for parameter restricted applications.

AUROC Performance: For each set-up the best result is marked by an underline (Tables 1–3). To get an overview if GraN is always in the range of the best result we additionally marked the result of GraN in bold if its performance is less than two percentage points worse than the best result. Only two set-ups fell below the two percent mark for the adversarial, one for the real world corruptions and none for the out-of-distribution case. The last line in all tables provides information on the average performance of each method. On average GraN outperforms LID by 4.41 percentage points and Mahalanobis by 0.28 percentage points for the adversarial example setting. For the out-of-distribution setting GraN outperforms ODIN in average by 6.26 percentage points and Mahalanobis by 0.42 percentage points. The most significant performance difference is found for the real world set-ups. Here, GraN was on average over 15.76 percentage points better than the Mahalanobis method. The Baseline method is 13.06 percentage points worse than GraN, which confirms the relevance of the gradient analysis of GraN. The Mahalanobis method has problems with detection especially when DenseNet is used as the underlying classification network. The

Table 3
AUROC-score and efficiency comparison for fault leading data corruption detection. We thrive for a parameter and runtime optimized solution at the same time, hence we marked the results of the most efficient method GraN that are in the range of (less than 2 percentage points worse) the best result in **bold**. The best results are marked by an underline.

Net	Data	Corruption	Prepro. Baseline			Mahalanobis [13]			GraN [28]		
		Noise	Para.	Time	AUROC	Para.	Time	AUROC	Para.	Time	AUROC
DenseNet	CIFAR-10	Original	2	<0.01[s]	79.68	157.9k	0.07[s]	50.95	0.3k	0.04[s]	**87.74**
		Gaussian			82.23			57.89			**92.63**
		Shot			79.64			58.90			**91.11**
		Impulse			82.41			61.47			**91.96**
	CIFAR-100	Original	2	<0.01[s]	78.01	214.1k	0.07[s]	54.18	0.3k	0.04[s]	**85.16**
		Gaussian			74.21			59.33			**89.51**
		Shot			76.22			58.00			**89.03**
		Impulse			67.19			56.95			**85.52**
	SVHN	Original	2	<0.01[s]	58.62	157.9k	0.07[s]	90.40	0.3k	0.04[s]	84.88
		Gaussian			78.64			72.84			**90.26**
		Shot			78.28			71.01			**89.80**
		Impulse			73.25			73.16			**87.10**
ResNet	CIFAR-10	Original	2	<0.01[s]	84.90	362.5k	0.02[s]	94.69	0.1k	0.01[s]	**96.22**
		Gaussian			75.65			72.17			**87.66**
		Shot			74.09			75.58			**85.73**
		Impulse			82.17			84.30			**92.00**
	CIFAR-100	Original	2	<0.01[s]	76.73	454.7k	0.02[s]	84.72	0.1k	0.01[s]	**87.04**
		Gaussian			73.32			82.31			**87.96**
		Shot			73.16			83.69			**88.20**
		Impulse			67.65			83.45			**86.02**
	SVHN	Original	2	< 0.01[s]	61.02	362.5k	0.02[s]	91.65	0.1k	0.01[s]	**89.67**
		Gaussian			84.32			82.49			**92.47**
		Shot			84.98			81.21			**92.97**
		Impulse			81.15			81.27			**90.39**
	Average				76.14			73.44			**89.20**

direct connection of each layer to all previous layers seems to make an evaluation over the Mahalanobis distance difficult. GraN on the other hand seems to have no problems with the DenseNet set-ups. Also for the ResNet set-ups GraN performs on average significantly better than Mahlanobis. We interpret the advantage of GraN as follows: The real world corruption set-ups are build such that the datapoints lie on the classification boundary. Mahalanobis measures the layer-wise distance to correctly classified data. For data points that are on the border between being classified correctly or incorrectly, the layer-wise distance is very small. GraN on the other hand measures the contradiction within the network to the predicted class via the gradient, which is more meaningful for such data points.

These results in addition to the investigated parameter and runtime requirements show GraN in advantage especially for real world scenarios or applications with limited resources.

4.5. Pre-processing relevance study

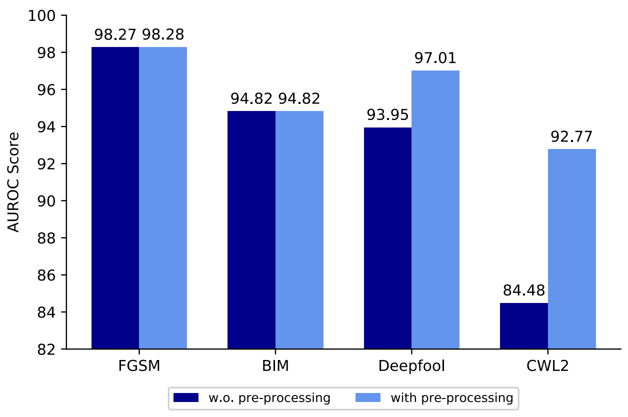
In order to investigate the relevance of the pre-processing procedure used in GraN we performed experiments without any pre-processing. In this experiments only the original image x is used for the algorithm and replaces the Gaussian smoothed image $\tilde{x}$. A direct performance comparison with and without pre-processing for an adversarial and an out-of-distribution setting is shown in Fig. 2.

For the out-of-distribution setting the pre-processing leads to a performance improvement but no tendency is visible if it is more relevant for some out-of-distribution data. For the adversarial setting however, such a trend is visible. Especially for the two attack methods Deepfool and CWL2, the pre-processing leads to a performance boost. When comparing the CWL2 and the Deepfool adversarial attack methods to those of FGSM and BIM we notice that Deepfool and CWL2 are more advanced methods. This means that the average euclidean difference between the original and a generated adversarial image that fools the DNN is typically smaller. The Gaussian pre-processing procedure is able to denoise the image from a small amount of noise. To remove a high level of noise, a high smoothing rate would have to be chosen which then also destroys necessary image information. Therefore, the smoothing is not as beneficial for a high noise level. However, through that higher noise-level, FGSM and BIM are easier to distinguish from images of the original dataset. The performance on those is better in general, cf. Table 1. The gaussian smoothing is originally thought to remove sensor noise, hence as expected the highest difference in performance can be observed for the corrupted set-ups, but also for the original set-up the performance is improved by the pre-processing step. Depending on the setup and whether the associated performance loss can be tolerated, it is an option to omit the pre-processing step to save runtime.

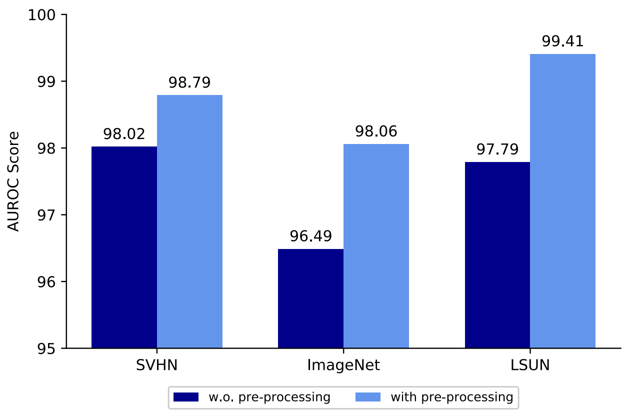
Inspired by the method ODIN and Mahalanobis we made some additional experiments replacing the Gaussian pre-processing procedure with the gradient descent based pre-processing procedure described in Eq. 1. However, it did not lead to an improvement of the detection score while consuming more runtime. Their gradient based pre-processings can be interpreted as a shift away from the original distribution and hence the generalization area, while the gaussian smoothing is thought to shift the data point closer to a correct classified in-distribution sample. Such a pre-processing is therefore not recommended.

4.6. Logistic regression study

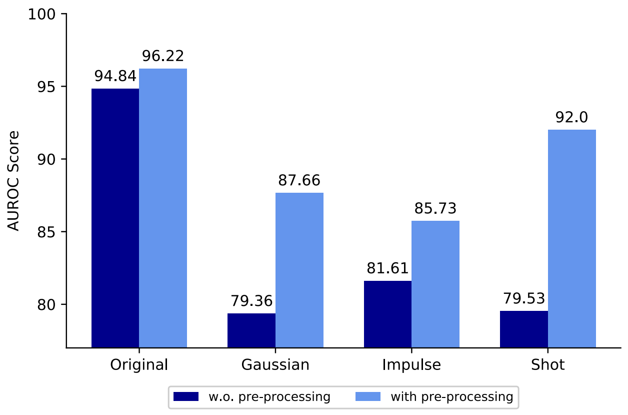
A logistic regression network can be seen as a one-layer neural network, resulting in the question if the performance of GraN could be improved by replacing the logistic regression part with a several



(a) Adversarial example detection for ResNet on SVHN.



(b) Out-of-distribution detection for DenseNet on CIFAR-100.



(c) Corruption detection for ResNet on CIFAR-10.

Fig. 2. Performance comparison for GraN with and without pre-processing evaluated for different networks and datasets.

layer neural network. We made some experiments on that using simple two- and three-layer neural networks but no relevant improvement in the detection performance could be observed. This indicates that an additional combination of features allowed by a deeper neural network does not bring any new information gain for the detection of a misclassification. The simple use of regression weighting is sufficient. Therefore, and since a logistic regression network can be trained hyper-parameter independently, we do not recommend using a multilayer neural network instead.

4.7. Layer importance study

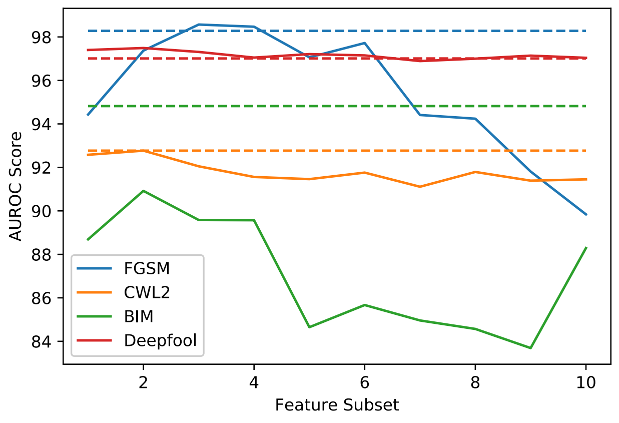
We investigate from which part of the network GraN extracts the most relevant features. Therefore, the feature vector is split into ten parts. Each part i consists of the i-th tenth of the feature vector and hence includes the gradient information of the corresponding layers of the classification neural network. In each set-up only the i-th part of the feature vector is used to train and evaluate GraN. The performance for all ten parts are compared. Examples for different settings and set-ups are shown in Fig. 3. The corresponding dashed line in the same color for each set-up shows the original performance when all features are used.

There is no clear tendency visible which subset performs best, the scores behave differently for each set-up. Some reach the baseline performance for almost all subsets, some get close for only one or two sub-sets and others are far below the baseline performance for all parts. Depending on the outlier data and the corresponding set-up this knowledge can be used to further save runtime for applications that are based on set-ups that already lead to good results when GraN is only evaluated on the gradients of the last layers. The backpropagation computations could stop when all relevant gradient information are extracted and the remaining runtime could be saved.

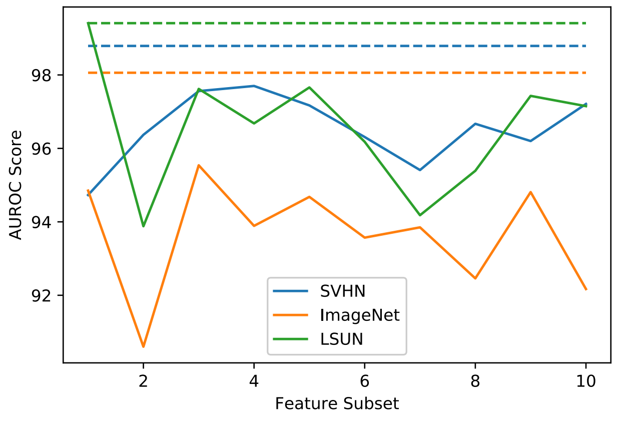
5. Conclusion

Choosing a method to detect misclassified samples led to a trade-off between state-of-the-art performance and approaches that are runtime and parameter efficient. This was problematic for applications in which the resources are limited and hence a runtime and parameter efficient method was required. Furthermore, state-of-the-art methods concentrated on either the detection of out-of-distribution data or adversarial examples, whereas no attention was paid to real world corruptions leading to misclassifications. In this paper, we therefore investigated the performance and runtime and parameter requirements of GraN, a detection method for misclassified samples. It achieves performance similar to state-of-the-art for the detection of adversarial and out-of-distribution samples while being runtime and parameter efficient. The main advantage of GraN however is its detection of real world corruptions that lead to misclassification. This was proven for various datasets, attack methods and network architectures in the field. Furthermore, we investigated the relevance of the pre-processing step of GraN, which leads to a performance boost especially for the detection of more advanced adversarial attacks and some real world set-ups. This shows that using GraN without pre-processing could be a solution for applications that do not expect advanced adversarial attacks and tolerate a small performance loss in order to further downsize runtime requirements. Experiments for layer importance show that further runtime savings, also associated with a performance loss, can be achieved by using only gradient information of the last layers. Thus, no full backpropagation step would have to be executed, which saves even more computing time.

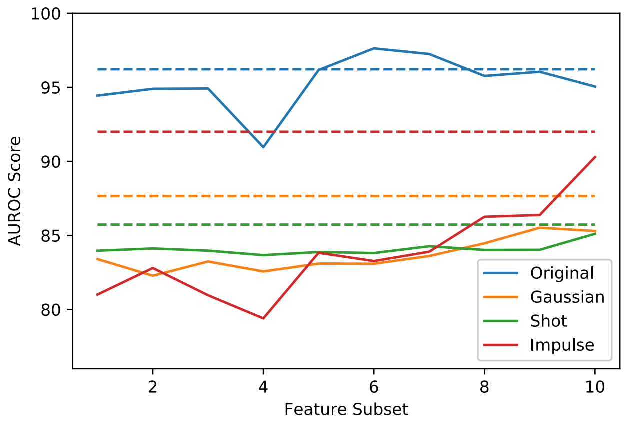
GraN needs examples of correctly and misclassified samples of the DNN for its training process. Such samples are usually a by-product of the training and validation process of any classification task anyway. However, a challenge to address in future work might be to adapt GraN to be independent of outlier data in its training procedure such that it generalizes well to unknown types of out-



(a) Adversarial example detection for ResNet on SVHN.



(b) Out-of-distribution detection for DenseNet on CIFAR-100.



(c) Corruption detection for ResNet on CIFAR-10.

Fig. 3. Performance comparison of GraN trained on different feature subsets. The i-th subset contains the i-th tenth of the feature vector. For all set-ups the dashed line in the corresponding color shows the original performance when the whole feature vector is used.

liers. One possible starting point could be to replace the norm-based feature extraction with a more sophisticated process.

CRediT authorship contribution statement

Julia Lust: Conceptualization, Methodology, Software, Writing - original draft, Visualization, Validation, Investigation. **Alexandru P. Condurache:** Conceptualization, Methodology, Investigation, Supervision, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] D.C. Ciresan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: 2012 IEEE Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, 2012, pp. 3642–3649, https://doi.org/10.1109/CVPR.2012.6248110.

[2] S. Ren, K. He, R.B. Girshick, J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks, in: C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7–12, 2015, Montreal, Quebec, Canada, 2015, pp. 91–99. http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks..

[3] Y. LeCun, Y. Bengio, G.E. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, https://doi.org/10.1038/nature14539.

[4] L. Gauerhof, P. Munk, S. Burton, Structuring validation targets of a machine learning function applied to automated driving, in: B. Gallina, A. Skavhaug, F. Bitsch (Eds.), Computer Safety, Reliability, and Security – 37th International Conference, SAFECOMP 2018, Västerås, Sweden, September 19–21, 2018, Proceedings, Vol. 11093 of Lecture Notes in Computer Science, Springer, 2018, pp. 45–58. doi:10.1007/978-3-319-99130-6_4. doi: 10.1007/978-3-319-99130-6_4..

[5] M. Karabatak, M.C. Ince, An expert system for detection of breast cancer based on association rules and neural network, Expert Syst. Appl. 36 (2) (2009) 3465–3469, https://doi.org/10.1016/j.eswa.2008.02.064.

[6] J. Lust, A.P. Condurache, A survey on assessing the generalization envelope of deep neural networks: predictive uncertainty, out-of-distribution and adversarial samples, CoRR abs/2008.09381. arXiv:2008.09381. URL https://arxiv.org/abs/2008.09381.

[7] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings, 2015. http://arxiv.org/abs/1412.6572..

[8] A. Kurakin, I.J. Goodfellow, S. Bengio, Adversarial examples in the physical world, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Workshop Track Proceedings, OpenReview.net, 2017. https://openreview.net/forum?id=HJGU3Rodl..

[9] N. Papernot, P.D. McDaniel, S. Jha, M. Fredrikson, Z.B. Celik, A. Swami, The limitations of deep learning in adversarial settings, EuroSP (2016) 372–387, https://doi.org/10.1109/EuroSP.2016.36.

[10] N. Carlini, D.A. Wagner, Towards evaluating the robustness of neural networks, in: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, IEEE Computer Society, 2017, pp. 39–57. doi:10.1109/SP.2017.49. URL: https://doi.org/10.1109/SP.2017.49..

[11] D. Hendrycks, T. Dietterich, Benchmarking neural network robustness to common corruptions and perturbations, in: International Conference on Learning Representations (ICLR)..

[12] S. Liang, Y. Li, R. Srikant, Enhancing the reliability of out-of-distribution image detection in neural networks, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018. URL: https://openreview.net/forum?id=H1VGkIxRZ..

[13] K. Lee, K. Lee, H. Lee, J. Shin, A simple unified framework for detecting out-of-distribution samples and adversarial attacks, in: S. Bengio, H.M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3–8, 2018, Montréal, Canada, 2018, pp. 7167–7177. URL: http://papers.nips.cc/paper/7947-a-simple-unified-framework-for-detecting-out-of-distribution-samples-and-adversarial-attacks..

[14] Y. Hsu, Y. Shen, H. Jin, Z. Kira, Generalized ODIN: detecting out-of-distribution image without learning from out-of-distribution data, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13–19, 2020, IEEE, 2020, pp. 10948–10957. doi:10.1109/CVPR42600.2020.01096..

[15] P. Oberdiek, M. Rottmann, H. Gottschalk, Classification uncertainty of deep neural networks based on gradient information, in: L. Pancioni, F. Schwenker, E. Trentin (Eds.), Artificial Neural Networks in Pattern Recognition - 8th IAPR

TC3 Workshop, ANNPR 2018, Siena, Italy, September 19–21, 2018, Proceedings, Vol. 11081 of Lecture Notes in Computer Science, Springer, 2018, pp. 113–125. doi:10.1007/978-3-319-99978-4_9..

[16] 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, in: H.M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8–14, 2019, Vancouver, BC, Canada, 2019, pp. 14680–14691. URL: http://papers.nips.cc/paper/9611-likelihood-ratios-for-out-of-distribution-detection..

[17] Q. Yu, K. Aizawa, Unsupervised out-of-distribution detection by maximum classifier discrepancy, in: 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019, IEEE, 2019, pp. 9517–9525. doi:10.1109/ICCV.2019.00961..

[18] C.S. Sastry, S. Oore, Detecting out-of-distribution examples with gram matrices, in: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13–18 July 2020, Virtual Event, Vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 8491–8501. URL: http://proceedings.mlr.press/v119/sastry20a.html..

[19] E. Zisselman, A. Tamar, Deep residual flow for out of distribution detection, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13–19, 2020, IEEE, 2020, pp. 13991–14000. doi:10.1109/CVPR42600.2020.01401..

[20] A. Vyas, N. Jammalamadaka, X. Zhu, D. Das, B. Kaul, T.L. Willke, Out-of-distribution detection using an ensemble of self supervised leave-out classifiers, in: V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss (Eds.), Computer Vision - ECCV 2018–15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part VIII, Vol. 11212 of Lecture Notes in Computer Science, Springer, 2018, pp. 560–574. doi:10.1007/978-3-030-01237-3_34..

[21] X. Ma, B. Li, Y. Wang, S.M. Erfani, S.N.R. Wijewickrema, G. Schoenebeck, D. Song, M.E. Houle, J. Bailey, Characterizing adversarial subspaces using local intrinsic dimensionality, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018. URL: https://openreview.net/forum?id=B1gJ1L2aW..

[22] D. Meng, H. Chen, Magnet: a two-pronged defense against adversarial examples, in: CCS, 2017, pp. 135–147.

[23] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, J. Zhu, Defense against adversarial attacks using high-level representation guided denoiser, in: CVPR, 2018, pp. 1778–1787..

[24] W. Xu, D. Evans, Y. Qi, Feature squeezing: Detecting adversarial examples in deep neural networks, in: NDSS, 2018..

[25] S. Ma, Y. Liu, G. Tao, W. Lee, X. Zhang, NIC: detecting adversarial samples with neural network invariant checking, in: NDSS, 2019..

[26] L. Enderich, F. Timm, W. Burgard, SYMOG: learning symmetric mixture of gaussian modes for improved fixed-point quantization, Neurocomputing 416 (2020) 310–315, https://doi.org/10.1016/j.neucom.2019.11.114.

[27] C. Schorn, L. Gauerhof, Facer: A universal framework for detecting anomalous operation of deep neural networks, in: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), 2020, pp. 1–6, https://doi.org/10.1109/ITSC45102.2020.9294226.

[28] J. Lust, A.P. Condurache, Gran: An efficient gradient-norm based detector for adversarial and misclassified examples, arXiv preprint arXiv:2004.09179. https://arxiv.org/abs/2004.09179..

[29] F. Crecchi, M. Melis, A. Sotgiu, D. Bacciu, B. Biggio, Fader: Fast adversarial example rejection, arXiv preprint arXiv:2010.09119..

[30] R.M. Haralick, L.G. Shapiro, Computer and Robot Vision, vol. 1, Addison-wesley Reading, 1992.

[31] G. Huang, Z. Liu, L. van der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21–26, 2017, IEEE Computer Society, 2017, pp. 2261–2269. doi:10.1109/CVPR.2017.243..

[32] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, IEEE Computer Society, 2016, pp. 770–778. doi:10.1109/CVPR.2016.90..

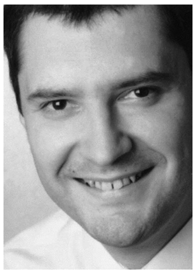
[33] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, Tech. rep., Citeseer (2009). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf.

[34] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A.Y. Ng, Reading digits in natural images with unsupervised feature learning, in: NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011. URL: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

[35] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, Li Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255. doi:10.1109/CVPR.2009.5206848..

[36] F. Yu, Y. Zhang, S. Song, A. Seff, J. Xiao, LSUN: construction of a large-scale image dataset using deep learning with humans in the loop, CoRR abs/1506.03365. arXiv:1506.03365. URL: http://arxiv.org/abs/1506.03365..

Julia Lust studied mathematics and received the BSc degree from the Rheinisch-Westfälische-Technische-Hochschule Aachen (RWTH), Germany. In 2016, she joined Robert Bosch GmbH for an internship in the department of Driver Assistance Systems. In 2019 she earned the MSc degree from the University Heidelberg, Germany. She then returned to Bosch and started her PhD at the department for Automated Driving in cooperation with the University of Lübeck. Her current research interest is assessing the generalization envelope of deep neural networks to predict possible failure of the learning method with a focus on autonomous driving applications.



Alexandru Paul Condurache received the Dipl.-Ing. degree in electrical engineering from the 'Politehnica' University of Bucharest, Romania in 2000. In 2007 he received the Dr.-Ing. degree and in 2014 the Dr.-Ing. Habilitation degree in computer science from the University of Luebeck, Germany. From 2002 to 2013, he was with the Institute for Signal Processing, University of Luebeck, as a Research Associate. Since 2013 he is with Robert Bosch GmbH. His current research interests include expressing and integrating prior knowledge in machine learning and investigating the generalization envelope of decision algorithms with a focus on autonomous driving applications.