Federal Office
for Information Security

Reliability Assessment of Traffic Sign Classifiers

Document history

Version	Date	Editor	Description
1.0	May 2020 – July 2020		Document created.

Authors: The report was prepared by LatticeFlow, a company founded by leading researchers from ETH Zürich with the goal to help AI teams build reliable AI systems. Authors of the report are the following members from the LatticeFlow team:

Pavol Bielik
Dr. Petar Tsankov
Prof. Andreas Krause
Prof. Martin Vechev

To contact the authors, please use contact@latticeflow.ai or <first name>@latticeflow.ai

The work was conducted in collaboration with the following members from the Bundesamt für Sicherheit in der Informationstechnik (BSI):

Dr. Christian Berghoff
Dr. Arndt von Twickel

Federal Office for Information Security
Post Box 20 03 63
D-53133 Bonn
E-Mail: <first name>.<last name>@bsi.bund.de
Internet: https://www.bsi.bund.de
© Federal Office for Information Security 2020

Executive Summary

Building reliable artificial intelligence (AI) systems requires systematic methods for assessing their quality to gain confidence in their correctness and identify deficiencies that AI teams must address. The purpose of this report is to evaluate how state-of-the-art techniques for testing neural networks can be used to assess neural networks, identify their failure modes, and gain insights on how to improve them. The work was conducted by LatticeFlow and was commissioned by Germany's federal office for information security.

In our study, we consider two neural networks that classify images of traffic signs into their corresponding classes (e.g., stop, priority, etc.). We use the LatticeFlow platform to assess whether the neural networks reliably recognize traffic signs under different environments that arise in practice. Concretely, we test the two neural networks against common environmental conditions supported by LatticeFlow, such as changes to the lighting, and custom ones specific to the domain of traffic signs, such as placing stickers on the sign surface. We report on our key insights gained in the process and how they can be used to improve the neural networks:

1. **Neural networks can be reliable for some environmental conditions.** We can specify important environmental conditions formally and systematically test the neural networks against them. The evaluated neural networks are reliable against some basic conditions, such as camera rotation.
2. **At the same time, neural networks are unreliable for others.** The neural networks can be unreliable against some basic conditions, such as image scaling and brightness changes. These identify possible problems with the neural networks and suggest where the effort needs to be spent on improving them in the future.
3. **It is critical to test against domain-specific environmental conditions.** The neural networks are highly unreliable against domain-specific conditions, such as placing traffic sign stickers. It is, therefore, important to define and consider these both during testing and training.
4. **Testing against combinations of environmental conditions is necessary.** Multiple environmental conditions can occur simultaneously, such as image scaling and blurring, and this may lead to additional failures not uncovered when testing the networks against these individually. Concretely, the failure rate of the neural networks doubles when considering the combination of these conditions.
5. **Test results can help elicit important failure modes.** The test process reveals inputs for which the neural networks' predictions are brittle. Clustering these inputs helps pinpoint important failure modes, such as traffic signs exposed to direct sunlight or traffic signs obstructed by objects.

Finally, we note that while our study focuses on testing traffic sign classifiers, the presented concepts and findings are general and can be used to test neural networks trained on different datasets (such as medical datasets) and other computer vision tasks (such as object detection and segmentation).

Zusammenfassung

Die Entwicklung zuverlässiger Systeme auf Basis künstlicher Intelligenz (KI) erfordert systematische Methoden zur Bewertung ihrer Qualität, um Vertrauen in ihre Korrektheit zu gewinnen und Mängel zu identifizieren, die behoben werden müssen. Dieser Bericht hat zum Ziel, zu evaluieren, wie aktuell verfügbare, den Stand der Technik widerspiegelnde Methoden zum Testen neuronaler Netzwerke genutzt werden können, um diese zu bewerten, ihre Schadensmodi zu identifizieren sowie Verbesserungsmöglichkeiten zu erkennen. Diese Arbeit wurde von der Firma LatticeFlow durchgeführt und vom Bundesamt für Sicherheit in der Informationstechnik (BSI) beauftragt.

Die Studie betrachtet zwei neuronale Netze, die Bilder von Verkehrsschildern klassifizieren (z. B. Stoppschild, Vorfahrtschild etc.). Die Studie nutzt die Plattform von LatticeFlow, um zu beurteilen, ob die neuronalen Netze Verkehrsschilder in verschiedenen Umgebungen, die in der Praxis auftreten, zuverlässig erkennen. Konkret testen wir die beiden neuronalen Netze gegenüber allgemeinen Umgebungsbedingungen, die von LatticeFlow unterstützt werden, wie Änderungen der Lichtverhältnisse, sowie gegenüber Veränderungen, die für den Bereich der Verkehrsschilder spezifisch sind, wie das Aufkleben von Stickern auf die Schilder. Dabei wurden die folgenden Erkenntnisse gewonnen:

1. **Neuronale Netze können unter einigen Umgebungsbedingungen zuverlässig funktionieren.** Wichtige Umgebungsbedingungen können formal spezifiziert und die Netze systematisch dagegen getestet werden. Die untersuchten neuronalen Netze funktionieren zuverlässig unter einigen grundlegenden Bedingungen, wie einer Rotation der Kamera.
2. **Gleichzeitig sind neuronale Netze unter anderen Bedingungen unzuverlässig.** Die neuronalen Netze können unter einigen grundlegenden Bedingungen, wie Skalierung und Änderung der Helligkeit, unzuverlässig sein. Hierdurch lassen sich mögliche Probleme der neuronalen Netze identifizieren und Erkenntnisse gewinnen, an welcher Stelle zukünftige Anstrengungen nötig sind, um sie zu verbessern.
3. **Es ist entscheidend, Tests unter anwendungsspezifischen Umgebungsbedingungen durchzuführen.** Die neuronalen Netze weisen unter anwendungsspezifischen Bedingungen, wie dem Aufkleben von Stickern auf die Schilder, eine hohe Unzuverlässigkeit auf. Es ist daher wichtig, diese sowohl während des Testens als auch während des Trainings zu definieren und zu berücksichtigen.
4. **Es ist notwendig, Tests unter Kombinationen von Umgebungsbedingungen durchzuführen.** Mehrere Umgebungsbedingungen, wie Skalierung und Unschärfe, können gleichzeitig auftreten. Dies kann zu zusätzlichen Fehlern führen, die nicht erkannt werden, wenn die Netze einzeln unter diesen Bedingungen getestet werden. In einem konkreten Beispiel verdoppelt sich die Fehlerrate der Netze, wenn eine Kombination dieser beiden Bedingungen (Skalierung und Unschärfe) betrachtet wird.
5. **Die Testergebnisse können genutzt werden, um wichtige Schadensmodi zu eruieren.** Der Testvorgang zeigt auf, für welche Eingaben die Vorhersagen der neuronalen Netze wenig robust sind. Wenn diese Eingaben zu Clustern angeordnet werden, können damit wichtige Schadensmodi, wie direkte Sonneneinstrahlung oder die Verdeckung der Verkehrsschilder durch andere Objekte, bestimmt werden.

Abschließend ist festzustellen, dass der Fokus dieser Studie zwar auf dem Testen von Netzen zur Verkehrsschildklassifizierung liegt, die vorgestellten Konzepte und Erkenntnisse jedoch allgemeiner Natur sind und zum Testen von neuronalen Netzen verwendet werden können, die auf andersartigen Datensätzen (wie medizinischen Datensätzen) oder für andere Aufgaben in der Bilderkennung (wie die Objekterkennung und die Segmentierung) trainiert wurden.

Contents

1 Introduction

Over the last few years, deep neural networks have been established as the state-of-the-art technology for building visual perception systems. In turn, they have become the core building block for developing systems for disease prediction [9, 4, 10], fault detection [8], quality inspection [6], autonomous driving [11, 3, 12] and many others. One of the main tasks and challenges in deploying such practical artificial intelligence systems is building *reliable* deep neural networks, i.e., models that achieve high accuracy and work reliably under a diverse set of environments that may occur in practice.

An essential prerequisite to building reliable neural networks is the ability to thoroughly assess the trained model's quality and systematically identify any failure modes, i.e., conditions that cause the model to fail. Without this, it is impossible to gain confidence in the model's quality and understand how to improve it. To illustrate, consider building a model that classifies images of traffic signs into their corresponding classes (stop, priority, etc.). It is evident that, to operate reliably, the model must classify traffic signs correctly under different lighting environments, camera angles, and so forth, which naturally occur in the target environment. Moreover, suppose that the model is prone to systematic failures in scenarios where the traffic sign is partially occluded by objects, such as tree branches. It is then imperative to find these failure modes and improve the model based on this knowledge, e.g., by suitably enhancing the training dataset.

Unfortunately, the current practice of training and testing neural networks does not facilitate finding common failure modes of the trained model. After a model has been trained, one typically evaluates its quality by measuring its accuracy, i.e., the fraction of correctly predicted test inputs. However, this metric does not provide any assurance that the model works reliably under certain conditions (e.g., camera noise), nor does it bring any insights into what causes the model to fail. To cope with this, currently, machine learning teams tend to manually inspect the model's behavior on different test inputs to identify deficiencies and understand how to enhance it. Instead, as the data and the environment are continuously changing, it is crucial that such failure modes are identified automatically by the system and integrated directly into the development process early on.

Objectives The objective of this project is to demonstrate how state-of-the-art robustness testing techniques can be used to:

1. Assess the robustness of the neural network with respect to important conditions that may occur in practice (rotations, brightness changes, etc.);
2. Uncover important failure modes of the neural network, such as common types of inputs that cause the neural network to mispredict;
3. Make informed decisions in the design process by comparing the performance of different neural network architectures;
4. Understand how to improve the neural network by enhancing the quality of the training dataset based on identified failure modes.

The results aim to highlight the benefits and limitations of robustness testing techniques in the quality assurance process of neural networks.

Reliability assessment of traffic sign classifiers As concrete test targets for our experiments, we consider two neural networks that classify traffic signs. Both networks have been selected and provided by the BSI. We show the key steps of the reliability assessment in Figure 1. The input to the testing process consists of the trained neural network and the test dataset. Also, the testing platform is configured with *robustness properties*, which capture a diverse set of conditions that may occur either naturally or adversarially in the target environment. Importantly, these properties enable the testing platform to check how well the network handles important cases, such as changing the input image's orientation, adjusting colors, placing stickers, etc. In practice, the properties are derived from the functional requirements of the target system.

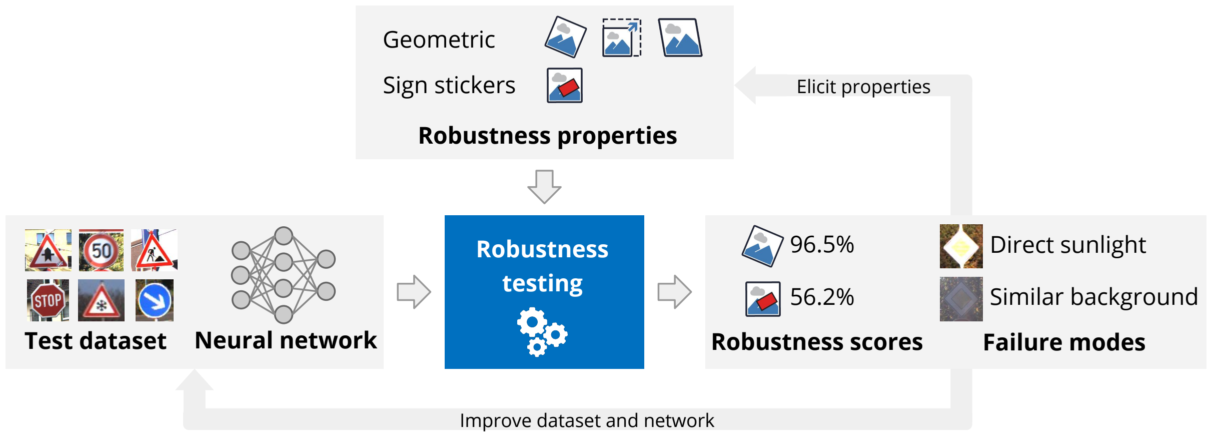


Figure 1: Reliability assessment of neural networks. The testing platform takes as input a neural network and its test dataset, along with robustness properties that capture important environmental conditions (camera position, sign stickers, etc.), and assesses the neural network against these together with its common failure modes.

Based on the inputs, we use LatticeFlow[1], a state-of-the-art platform for testing neural networks, to automatically assess the neural network's behavior and identify failures. Concretely, the platform reports the neural network's robustness to the different scenarios captured by the robustness properties. Based on these results, we also demonstrate how one can pinpoint common failure modes. For example, we show how one can discover specific (combinations of) transformations that cause the network to misbehave. Further, we show how to find test inputs for which the model is highly uncertain (relative to the properties). This information is essential to understand how to enhance the dataset and define additional custom properties, which improve both the neural network and the testing process.

We note that while the report focuses on the task of classifying traffic signs, the presented concepts are general and can be used to test neural networks trained on different datasets (e.g., medical images) and other computer vision tasks (e.g., object detection, segmentation).

Roadmap The rest of the report is organized as follows. In Section 2, we provide details on the datasets and neural networks used in our experiments. In Section 3, we describe the key steps of our robustness assessment experiments, and in Section 4, we highlight our results. Finally, in Section 5, we draw conclusions. All technical details, including the formalization of the considered robustness properties and detailed experimental results, are provided in the extended version of the report [13].

[1]For more information, visit https://latticeflow.ai.

2 Case Study: Traffic Sign Classifiers

We now present details about our case study on the reliability assessment of traffic sign classifiers.

Inputs To perform the assessment, we use two inputs:

- *Test Dataset* consisting of images with the corresponding ground-truth labels.
- *Trained Models* that perform a given task, in our case classification of traffic signs.

We note that while we focus on a given task, our methodology is general and independent of the actual task. It is parameterized only by the trained models, test dataset, and robustness specification (discussed in Sections 4.1 and 4.2). In particular, the robustness analysis considered in our case study can be applied to computer vision models trained on different datasets (e.g., medical imaging) and models that implement various tasks (e.g., object detection or segmentation).

2.1 Dataset: German Traffic Signs (GTSRB)

As our main dataset, we use the test partition of the German Traffic Sign Recognition Benchmark dataset [22] (GTSRB). This dataset consists of $12,630$ color images of German road signs, classified into 43 classes. The dataset images have different resolutions, with height and width ranging between $25-266$ and $25-232$ pixels, respectively. In Figure 2, we show one example image for each of the 43 traffic sign classes present in the dataset. This dataset was chosen by the BSI.

Additionally, we test the models using the DFG traffic sign dataset [28], which consists of 200 traffic sign categories collected in Slovenia. We manually created a mapping to the classes that are also represented in the GTSRB dataset and removed the rest (we provide full details of the preprocessing used in [13], Appendix E).

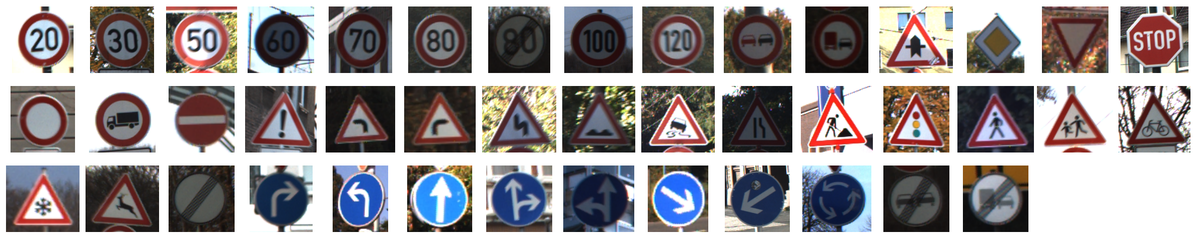


Figure 2: Example traffic signs drawn from the GTSRB dataset [22]. We show one example from each of the 43 traffic sign classes.

2.2 Trained models

We test two neural networks prepared and supplied by the BSI team:

1. The first neural network, called **pre-trained**, has 99.0% accuracy. It uses the pre-trained Inception-v3 model [27], which is trained on the ImageNet dataset [17] and fine-tuned for the GTSRB dataset.
2. The second one, called **self-trained**, has 97.4% accuracy. It uses an architecture based on Inception-v3 but with reduced size and without pre-training.

Both networks were trained with a data augmentation policy that applies the following random transformations: scaling using a factor between 0.6 and 1.0, rotation by an angle within 15 degrees, color changes – brightness, contrast, saturation, and hue – by a factor of up to 0.1, and random change to a grayscale format. The transformed images are then scaled to the neural network's expected resolution – 32×32 pixels for the self-trained network and 299×299 for the pre-trained network.

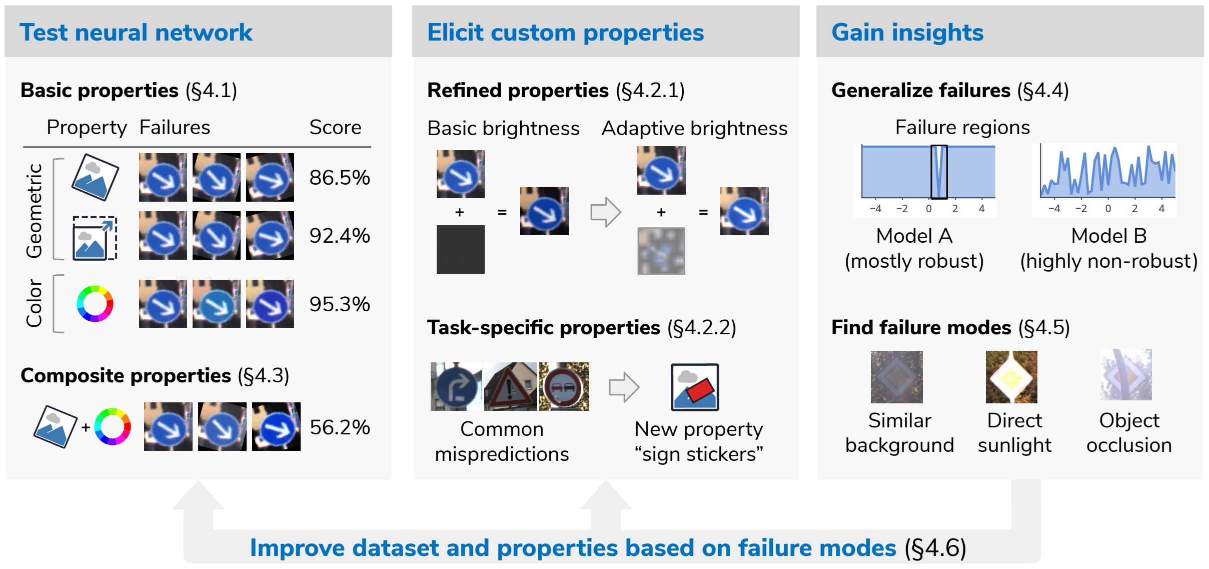


Figure 3: High-level steps of the testing process. First, we test the model against a set of basic robustness properties and their combinations. Based on the results, we elicit stronger properties (e.g., adaptively changing brightness for different parts of the image) and task-specific properties, such as adding stickers. Next, the results are analyzed to provide insights by identifying common failure modes. The failure modes are then used to iteratively improve both the model, the dataset as well as to elicit new robustness properties.

3 Reliability Assessment Overview

We now describe the high-level steps of our testing approach (depicted in Figure 3).

3.1 Test the neural network

First, we test the neural network against generic robustness properties (see the left box in Figure 3). The generic robustness properties apply to a wide range of computer vision models, and include different types of geometric transformations (e.g., rotation, scaling, perspective), change to image colors (e.g., adjust hue, contrast, brightness), natural noise (e.g., sensor failures), perturbations to individual image pixels, and others. Formally, each robustness property transforms an existing image into a new image according to the property (e.g., rotation turns the image by a given degree). To preserve the original image label, all transformations have bounds, which define the maximum amount of transformation applied to the original image (such as the maximum degree for rotation).

Robustness score We test the neural network against the individual robustness properties, and report each property's robustness score. The robustness score is defined as the fraction of robust samples in the dataset. We say that a sample is robust to a given property if the network produces the correct label for all transformations captured by the property. For example, suppose the model achieves 86.5% robustness with respect to the property stating that the model must be robust to 15 degree rotations. This indicates that for 13.5% of the samples, we found a rotation of the original image within 15 degrees which causes the model to predict the wrong label. We note that finding such failure inputs is challenging due to the extremely large space of possible transformations captured by each robustness property. Further, we note that robustness is defined only over samples for which the model already predicts the correct label as all incorrect samples are trivially non-robust.

Testing robustness property combinations In addition to testing the neural network against individual robustness properties, we also consider combinations of robustness properties, such as rotations followed by adding noise. Testing such combinations is important because even if the network is robust to the individual robustness properties, it may break for their combination. Our experiments, presented in Section 4.3, show that combining properties can indeed result in significantly more failures.

3.2 Elicit stronger and task-specific robustness properties

Having tested the network against generic robustness properties, we define additional properties that cover more advanced and task-specific robustness properties as a next step.

Stronger task-agnostic properties While the generic robustness properties are useful to discover common model failures, they are not powerful enough to model many real-world scenarios. For example, the basic brightness property uniformly changes the brightness of the entire image. To refine it, we define adaptive brightness, which applies local brightness changes to the image (see Figure 3). Note that, compared to basic brightness, adaptive brightness models strictly more failures. We present such stronger robustness properties in Section 4.2.1.

Task-specific robustness properties In addition, to strengthening the basic generic properties, we also define custom, task-specific robustness properties, which target specific computer vision tasks, such as processing traffic signs or MRI scans. For example, because people often place stickers on traffic signs, we explicitly define a "traffic sign stickers" property, which enables us to test the robustness of traffic sign classifiers against sign stickers. To derive such custom properties, we inspect common failures over the dataset, indicating difficult-to-learn scenarios, and define properties that model these cases. We discuss additional robustness properties specific to traffic sign classifiers in Section 4.2.2.

3.3 Assess test results and gain insights

The last step of the testing process is to assess the test results and gain insights into the model's failure modes. Concretely, we inspect breakdowns of the test results across (i) the parameter space of the robustness properties and (ii) the test inputs. Both breakdowns are essential to observe patterns of systematic failures across different types of transformations and inputs.

Robustness across the parameter space Recall that each robustness property has parameters (such as the angle of rotation). The robustness score (defined in Section 3.1) aggregates the failures across all robustness parameters and hides how many parameters, and which ones, result in failures. To gain deeper insights into the failure modes, we plot the robustness across the parameter space, i.e., where the x axis is the parameter space (see top right of Figure 3). These plots are helpful to:

1. **Pinpoint failure regions**. For example, suppose the robustness score for rotation and brightness is low, indicating that the model is brittle for this combination of transformations. The breakdown may reveal the particular failure region, such as: "rotations by $2-4$ degrees and brightness increase by $5-10$% often result in failures".
2. **Compare different models**. For example, in Figure 3 we plot the robustness landscape of two models A and B. While both models are not robust to the considered robustness property, by considering the robustness landscape, we can see that model A is, in fact, significantly more robust than model B, because the latter fails for most values of the parameter space.

Find failure modes The second important breakdown is to plot the model's robustness across the test inputs. This helps identify inputs where the model is brittle and uncertain (i.e., inputs for which most robustness transformations result in a failure). For example, in Figure 3, we show three test inputs for which the model is brittle. These inputs suggest that the model is uncertain when classifying the priority road sign in conditions where the background is similar, when there is direct sunlight at the traffic sign, and when the sign is occluded by an object (such as a tree branch).

Finally, we note that finding failure modes is essential for training an accurate and robust models. Technically, this is often achieved by producing better datasets and eliciting additional robustness properties, both guided by the knowledge of failure modes. We discuss this point further in Section 4.6.

4 Main Results

In this section, we describe the main results of our study. To simplify the presentation, we structure our sections along four main dimensions:

- **Properties (§4.1-§4.2)** - we start by testing the models against basic generic robustness properties. Then, we refine these into stronger properties and design task-specific properties tailored to traffic sign classifiers.
- **Dependence (§4.3)** - we start by considering the robustness properties individually. Then, we demonstrate that it is critical to consider the properties jointly.
- **Metric (§4.4-§4.5)** - we begin by reporting a robustness score (a single number). Then, we show the benefits of reporting the robustness landscape (a plot).
- **Dataset (§4.6)** - initially, we use the original test dataset for our robustness analysis. Then, we design new synthetic datasets based on the failure modes discovered using our robustness analysis.

4.1 Testing against basic robustness properties

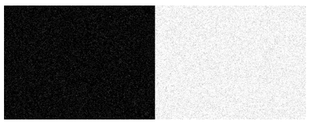
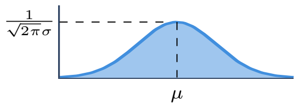
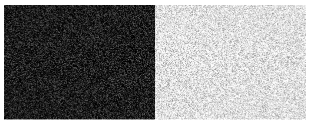
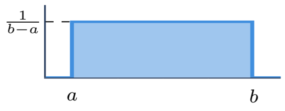
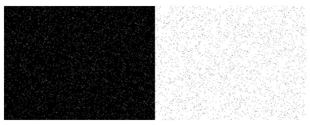
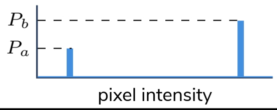
DATASET		PROPERTIES		DEPENDENCE		METRIC
Fixed	+	**Basic**	+	**Independent**	+	**Number**

As the first step, we test the network against a set of generic robustness properties applicable to a wide range of tasks that take images as inputs. These include properties that model natural image noise, perturb individual image pixels, perform geometric transformations, change the image colors, and general transformations such as image compression. If any of these properties is non-robust, it is a sign that a model deficiency has been found. However, as we will see later, such analysis is insufficient and care has to be taken when interpreting the results – high robustness of the model to a basic property checked in isolation (i.e., independently of other properties) does not mean the model is robust to combinations of multiple basic properties.

Given the large number of properties, and to avoid clutter, in what follows, we give an overview of the robustness for selected properties from each category. Further, note that we report *adversarial robustness*, which empirically checks whether the input image is robust to *all* transformations defined by the property (e.g., all image rotations). We provide the full results, including the formal specification of each property, in the extended version of this report [13].

Image noise As cameras are physical devices, there are many sources of noise that affect the resulting digital image, as illustrated in Table 1. Noise can occur naturally, such as Gaussian noise, which arises during image acquisition (e.g., due to poor illumination, high sensor temperature, or electronic circuit noise), it can be induced by sensors (e.g., photon-counting noise and speckle [20, 23]), it can be caused by the signal transmission (e.g., due to faulty switching, atmospheric disturbances), or arise as a result from various processing steps (e.g., due to analog-to-digital converters or quantization) [14].

Depending on the setting, we select a suitable noise model that accurately reproduces the spatial characteristics of the real-world noise source [15]. Then, we assess whether the neural network is robust to the specified type and amount of noise. As a concrete example, we illustrate the robustness of the two models to Gaussian noise in Figure 4. We see that both models are partially non-robust – the self-trained and the pre-trained models predict the wrong traffic sign for 4.5% and 1.2% of the images, respectively.

Noise Source	Noise Model	Visualization	Noise Distribution
Poor Illumination Sensor Temperature Circuit Noise	Gaussian Noise		
Quantization Analog-to-Digital Converter	Uniform Noise		
Faulty Switching Sensor Failure Transmission Loss	Impulse Noise[1]		

[1] also referred to as Salt and Pepper noise

Table 1: Examples of common noise sources for images, their corresponding noise model, visualization of the resulting noise, and the probability density distribution associated with the noise model.

Gaussian Noise Robustness

95.5%	**98.8%**
SELF-TRAINED	PRE-TRAINED

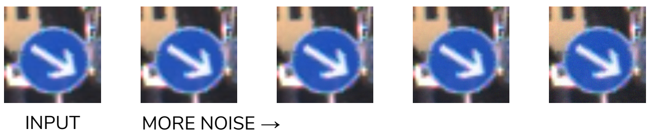


Figure 4: Robustness to Gaussian noise (generated with $\mu = 0$, $\sigma = 1$) together with an illustration of the noise applied to the input image. While the amount of noise added is imperceivable for humans, it is enough to cause the models to misclassify a significant amount of the samples.

Pixel perturbations For pixel perturbations, the robustness property is defined over individual pixels, each of which is allowed to be changed independently of other pixels. The only constraint is that the overall change to the image is below a specified threshold. Below we illustrate robustness in two settings that differ in how the allowed change to the original image is measured.

First, we allow modifying a fixed number of pixels by any amount, while the remaining pixels remain the same (referred to as L_0-norm). Note that even though both L_0-norm and Impulse noise define the same set of perturbations that can be applied to the image, pixel perturbations explore this space more effectively to find non-robust samples. In contrast, Impulse noise samples from the corresponding noise distribution. This difference can also be seen in the examples shown in Figure 5, which illustrates that, in some cases, it is enough to change a single pixel to break the model. When allowing changes of up to 32 pixels, the self-trained model robustness is only 56.9%. This is partially because the self-trained model's resolution is quite low at only 32×32 pixels. The robustness of the pre-trained model is significantly better, even when we allow changes for up to 128 pixels. However, given that the pre-trained model uses a much higher resolution of 299×299, the results are not directly comparable. We provide a thorough evaluation and description of how to make the results comparable between different models in our full report [13] (Appendix D).

Pixel L_0 Robustness

61.3%	**96.8%**
SELF-TRAINED	PRE-TRAINED

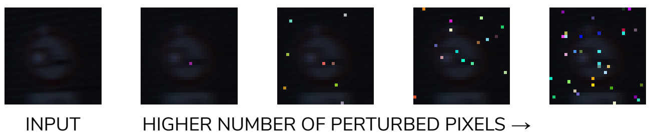


Figure 5: Robustness to pixel L_0 perturbations and visualizations of the non-robust images. Here, the number of perturbed pixels shown is 1, 8, 16 and 32.

As our second example, we allow all pixels to change, but by less than a specified threshold ϵ (referred to as L_∞-norm). The overview of the robustness in Figure 6 shows that both the self-trained and pre-trained models are highly non-robust. Here, the robustness of the pre-trained model is essentially zero, meaning that almost no images are classified correctly. This is even though the perturbation is still almost non-perceivable to humans.

Pixel L_∞ Robustness

53.5% SELF-TRAINED **2.1%** PRE-TRAINED

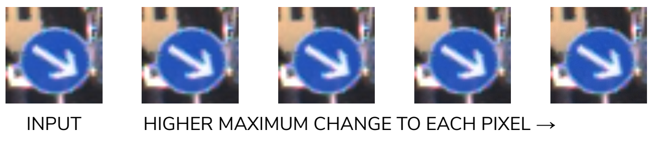


Figure 6: Robustness to L_∞ pixel perturbations with $\epsilon \leqslant 0.01$.

Geometric transformations The next set of robustness properties capture geometric transformations of the input image. The geometric transformations used are illustrated below and capture generic transformations that occur naturally in practice.

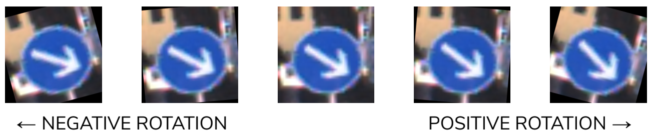
ROTATION TRANSLATION SCALING SHEARING BLURING SHARPENING FLIPPING PERSPECTIVE

For example, the scaling property corresponds to moving closer (or further away) from the traffic sign, and blurring captures the fact that the camera is typically moving. Rotation, translation, shearing, and perspective all encode different variations of the same image. We illustrate the robustness of two geometric properties (rotation and scaling) next and include the full results in [13] (Appendix B).

For rotation, both the self-trained and pre-trained models are relatively robust, containing only 2.4% and 0.5% non-robust samples, respectively. However, note that even such relatively high robustness still causes the error rate of both models to double. For scaling, the robustness is significantly worse, with 11.0% and 3.1% samples being misclassified, respectively. To understand the reasons behind the worse robustness of the scaling property (as well as other properties), we provide additional analysis in Section 4.5.

Rotation Robustness

97.6% SELF-TRAINED **99.5%** PRE-TRAINED



Scaling Robustness

89.0% SELF-TRAINED **96.9%** PRE-TRAINED

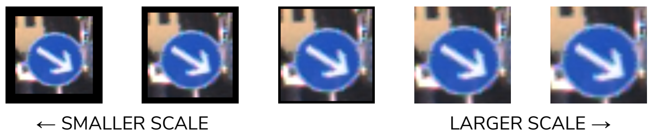


Figure 7: Overview of the robustness to rotation and scaling transformations. For rotation, the images were rotated by angles between -15 and 15 degrees. For scaling, we resized the images with a factor between 0.8 to 1.1 of the original size.

Color transformations As the name suggests, color transformations affect the image colors while keeping the other properties (e.g., rotation, perspective, noise) unchanged. The color properties are useful since, if they are strong enough, they allow modeling different environmental conditions under which the image was taken (e.g., amount of natural lighting) and differences between color post-processing performed on the image (either automatically by the camera or manually). The list of generic color properties evaluated in this work is shown below. We illustrate color robustness for two selected properties – brightness and hue – in Figure 9.

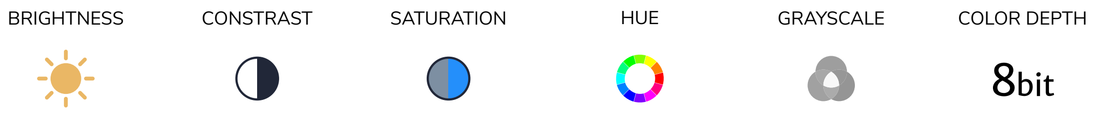


Figure 8: List of basic color properties evaluated in this work.

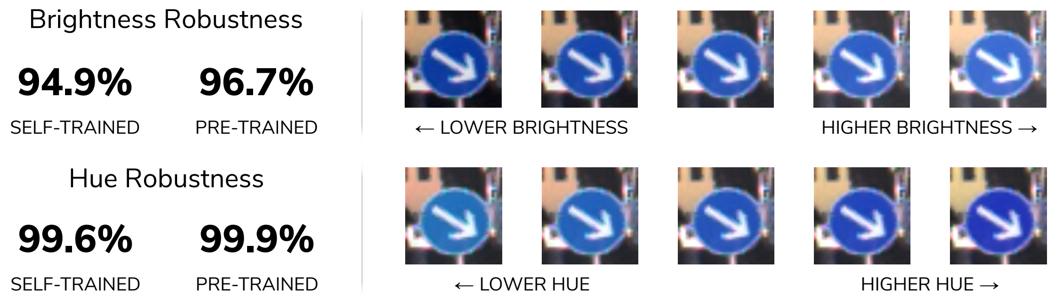


Figure 9: Robustness overview to brightness and hue changes. For both properties, the allowed change was in the range -0.1 to 0.1 of the original property value.

The results show the robustness to hue is exceptionally high, and less than 0.4% of samples are non-robust for both models. However, at the same time, both models are significantly less robust to brightness changes. Similarly to the geometric transformations, even though the robustness might appear quite high, the results show a higher misclassification rate than the model accuracy on the original test dataset. Concretely, for the brightness transformations, the error rates are $2\times$ and $4\times$ higher for the self-trained and pre-trained models, respectively.

General transformations The last set of robustness properties encode general image transformations. As an example in this category, we evaluate robustness to image compression, as shown in Figure 10. The robustness of both models is very high, with the self-trained model having 98.4% robustness and pre-trained model 98.7% robustness. Note that this is even though for the pre-trained model the compression level was set to very high. For the self-trained model, we used smaller compression levels as otherwise the compression artifacts dominate the image.

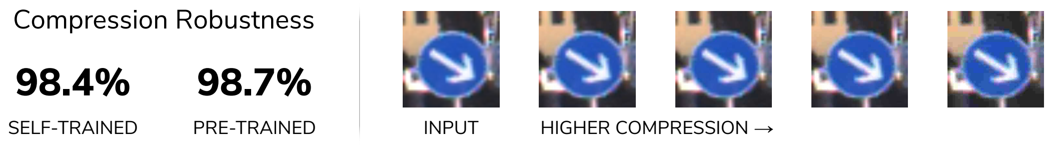
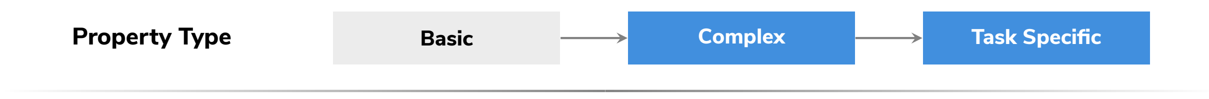


Figure 10: Robustness to using JPEG compression. For pre-trained model the images have resolution 299×299 and compression quality was set between 90 to 20. For self-trained model, the images are smaller with resolution only 32×32 and therefore we used higher compression quality from 90 to 80.

Summary In this section, we presented an overview of the basic assessment of two neural networks trained for the task of traffic sign classification. We have seen that while there are some properties for which the models are relatively robust (e.g., rotation, hue, compression), there are others where the models' robustness is significantly lower (e.g., scaling, brightness, pixel L_∞ perturbations). This is even though the set of perturbations were selected such that the transformed images are for many properties almost indistinguishable from the original image by humans. However, the results also show that if robustness is taken into account, it can be significantly improved (e.g., by suitable training or architecture selection). In particular, for properties that were included in the training (e.g., hue, brightness, rotation, translation), the robustness is relatively high. Further, there is also a notable difference between the two evaluated models, with a pre-trained model achieving higher robustness for all but one robustness property (while also achieving higher standard accuracy).

4.2 Eliciting stronger and task-specific properties



So far, we assessed the model using a set of basic robustness properties. While these are useful to discover issues, they are insufficient to assess the model against more advanced scenarios that occur in practice. To address this issue, we extend the robustness properties by developing stronger properties and task-specific properties tailored to traffic sign classification. This step is crucial for a more thorough assessment of the model. Indeed, our results show that the models have close to zero robustness against many of the stronger properties.

4.2.1 Stronger robustness properties

We illustrate two examples of stronger robustness properties – *adaptive color* transformations and *recolor* transformations. We selected color properties as for these, both the self-trained and pre-trained models achieved the highest robustness.

Adaptive colors The adaptive color transformations change different parts of the image differently. This is in contrast to the basic color properties from Section 4.1 that apply the same transformation for the whole image. As a concrete example, consider the images shown in Figure 11, which illustrates the difference between basic and adaptive brightness. Here, δ is used to visualize the brightness modification, which for the basic brightness simply makes the whole image darker (i.e., δ is a solid gray color). In contrast, adaptive brightness makes fine-grained changes, allowing some parts of the image to be brighter, and others darker. As a result, the model robustness to such stronger property is significantly lower – 37.4% for the self-trained model (from 94.9%) and 56.1% for the pre-trained model (from 96.7%). Note that while adaptive colors allow different changes to different parts of the image, they are selected to encourage consistency. That is, changes applied to spatially close locations are optimized to be similar to each other. For a formal definition of this property and the optimization procedure, please refer to [13] (Appendix A). Further, we note that this property applies to other color properties beyond brightness, as shown in Figure 12.

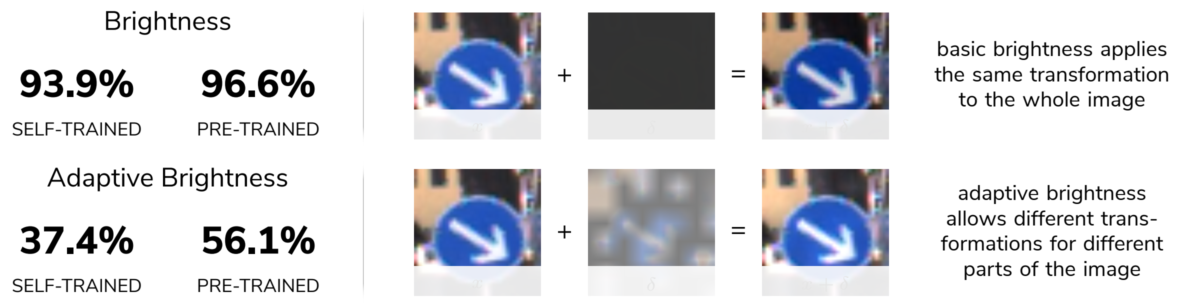

Brightness
93.9%
SELF-TRAINED
96.6%
PRE-TRAINED
Adaptive Brightness
37.4%
SELF-TRAINED
56.1%
PRE-TRAINED
+
=
basic brightness applies the same transformation to the whole image
adaptive brightness allows different transformations for different parts of the image

Figure 11: Robustness comparison of the basic and adaptive brightness property.

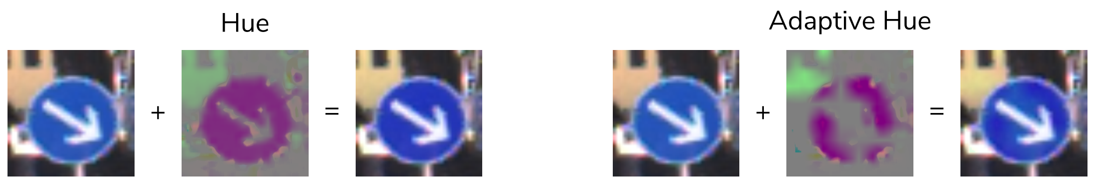


Figure 12: Illustration of the adaptive color applied to hue color property.

Recoloring The recoloring transformation [24] allows to change colors in the image directly, instead of changing the individual color properties (e.g., brightness, contrast, saturation, etc.). Given that the traffic sign's colors are an important feature for traffic sign classification, we restrict color changes to nearby colors. Overall, the robustness to recolor transformation is very low and significantly worse than any of the basic color transformations from Section 4.1. Concretely, the robustness of the self-trained network is only 41.6%, while the robustness of the pre-trained network is even lower at 39.9%.

4.2.2 Task-specific properties

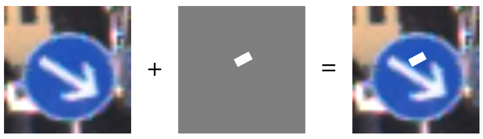
In addition to the generic image properties discussed so far, there are typically many properties that are specific for the given task at hand. Similar to the stronger robustness properties, we illustrate several task-specific robustness properties and show that the neural models can be highly non-robust to them. As a result, such task-specific properties must be part of the robustness evaluation.

Traffic sign stickers A typical example of a robustness property specific for the task of traffic sign classification is robustness to stickers placed on the surface of the traffic sign. Note that this is a property that is already included in the GTSRB dataset, which contains samples that contain stickers, e.g.:

We model this transformation by defining a robustness property that inserts a "sticker" on the traffic signs. We model stickers as white rectangles of varying size, orientation, and position on the traffic sign.

Traffic Sign Stickers

33.8%	27.2%
SELF-TRAINED	PRE-TRAINED



inserts a single sticker of varying position, size and orientation on the traffic sign

As the results show, the robustness of both the self-trained and the pre-trained models is extremely low. In fact, the models produce correct answers for only ≈ 30% of the images. This is even though we use relatively simple approximations of real-world stickers, which can be extended to different shapes and textures. Further, these transformations also translate to the physical world, as shown by [18].

Additional properties for traffic signs Naturally, there are many other task-specific properties that we can define. In Figure 13, we list more properties relevant for traffic sign classification and frequently occur in practice. Defining transformations that capture these properties is useful both for assessing the network robustness and generating a synthetic dataset used for training.

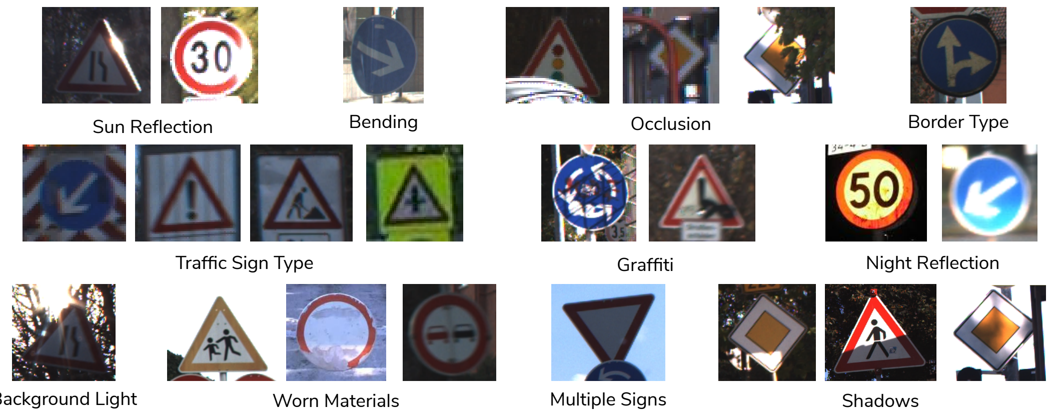


Figure 13: Real world examples of properties that are relevant for traffic sign classifiers.

4.3 Testing against multiple robustness properties

Property Dependence **Independent** → **Combinations** → **Ordering**

So far, the analysis performed in Sections 4.1 and 4.2 assumed that the robustness properties are independent and assessed the model to each property in isolation. However, the robustness properties can be composed together (e.g., combining brightness with rotation), effectively creating a new set of properties that should be assessed for robustness. The main technical difficulty in exploring the robustness properties jointly is that the resulting set is exponential in size. As a result, specialized techniques are required to explore this set efficiently (e.g., by identifying properties that are indeed dependent). In what follows, we illustrate the effect of considering the robustness properties jointly and include the full results in [13] (Appendix C).

Combining robustness properties We summarize the effect of considering the robustness properties jointly in Figure 14. Here, each point denotes a concrete combination of two robustness properties and the effect on the robustness. As a concrete example, let us consider the highlighted configuration in Figure 14 (left), which corresponds to combining the properties translate and scale. When considered independently of each other, the robustness to translation is 94.7%, and the robustness to scaling is 93.0%. However, when combined, the robustness decreases by additional 7.5% to 85.5%. We obtain similar results when evaluating the pre-trained model, as shown in Figure 14 (right). For both models, most property combinations decrease robustness. However, few combinations improve robustness (the points with positive robustness change in Figure 14). This result shows that combining all the properties together does not necessarily lead to a reliable model assessment. We note that this may be because the robustness assessment is phrased as an optimization problem, which becomes harder to solve when all the properties are combined.

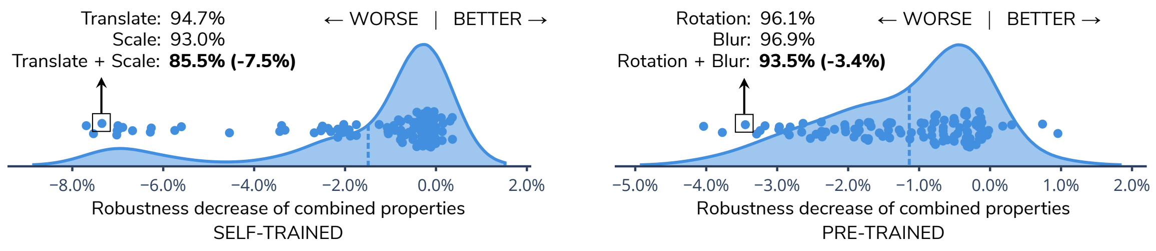


Figure 14: Effect of assessing model against property combinations for self-trained (left) and pre-trained (right) models. Each point corresponds to the relative change in robustness of a selected properties combination compared to considering the properties independently. For most properties, their combination decreases the robustness (shown as negative change).

Ordering robustness properties A natural question that arises when considering multiple properties is in which order they should be combined. While for some properties, the order leads to the same results (e.g., combining translation with brightness), for others, different orders lead to significantly different results. As a concrete example, consider combining scaling and blur property shown below:

property order can significantly affect the robustness Scale + Blur Robustness **81.4%** vs **87.6%** Blur + Scale Robustness

We can see that while the same two properties are used, the robustness of the pre-trained model (on the GTSRB dataset) to applying scaling first is almost twice as worse compared to applying scaling second.

4.4 Deeper insights by inspecting robustness plots

To evaluate model robustness, the typical practice is to report a single number that corresponds to the average per example robustness, as illustrated in Figure 15.

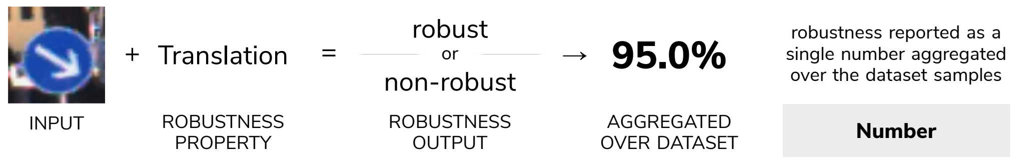


Figure 15: Assessing the model robustness by computing a single bit of information for each sample.

Here, for a given robustness property (e.g., translation), we compute whether each sample in the dataset is robust or not. While useful as a high-level assessment of the model's robustness, reporting a single bit of information for each image (robust/non-robust) is limiting for many reasons. First, it does not help understanding why the given sample is robust or non-robust (i.e., no explanation to support the decision is given). Second, it does not provide a measure of how confident the robustness assessment is. Finally, it does not allow fine-grained comparison between two samples (or models) that are both robust or non-robust. To address these limitations, instead of reporting a single number, we plot the robustness landscape, as shown in Figure 16.

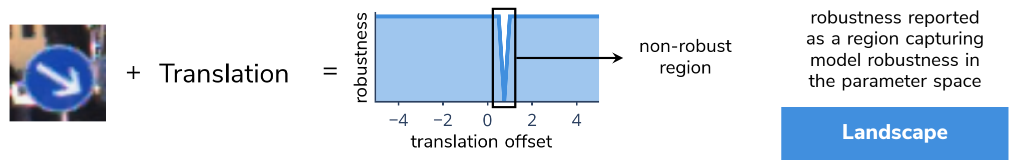

+ Translation =
robustness
−4
−2
0
2
4
translation offset
non-robust region
robustness reported as a region capturing model robustness in the parameter space
Landscape

Figure 16: Assessing the robustness using robustness landscape instead of a single number.

Here, for a given robustness property, we plot the model robustness across the parameter space of the given robustness property. For the translation property, the parameter space corresponds to different offsets with which the image can be shifted. As can be seen, even though the input image is non-robust, such analysis provides the explanation why – it is non-robust due to a small region when shifting the image by $+1$ pixel to the right. We can also see that apart from this region, the image is, in fact, highly robust for all other values in the parameter space.

Further, computing the robustness landscape also allows intuitive and more detailed comparison between multiple models (or samples) that are all robust (or non-robust). As a concrete example, consider three models shown in Figure 17 (the models A, B, C are only illustrative and do not correspond to the self-trained and pre-trained models).

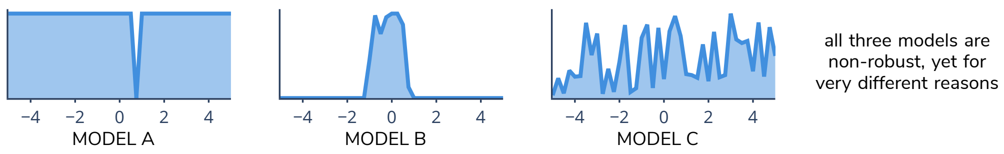


Figure 17: Comparing robustness landscapes of three hypothetical non-robust models.

Even though all the models are non-robust, they are non-robust for different reasons. Model A is non-robust due to a single model failure caused by shifting the image ≈ 1 pixel to the right. In contrast, model B is robust only to a small region around the original image and non-robust everywhere else. This may indicate that the model was not trained with robustness in mind, or that it was trained to be robust but only to the smaller region $(-1, 1)$.

4.5 Finding common failure modes

Dataset | Fixed → **Failure Modes** → Synthetic

In Section 4.4, we illustrated how providing a robustness landscape instead of a single number can help understand where the model is non-robust. In this section, we discuss how to gain insights into *why* and for which types of samples the model is non-robust. Concretely, we are interested in discovering failure modes of the model, that is, to find non-robust regions of the input space. For example, a failure mode for classifying priority road traffic signs ◈ is that the model is non-robust when the sign is under strong direct sunlight or when it is occluded with a different object (both illustrated in Figure 18). Being able to compute and analyze such failure modes brings many benefits:

- **Design new robustness properties** that capture transformations relevant to the failure modes. This is a data-guided approach to designing properties based on model failures, rather than based on the task description only, as discussed in Section 4.2.2.
- **Improve uncertainty estimates** as failure modes are a useful indicator of the model certainty.
- **Generate synthetic datasets** beyond what the robustness properties used so far can capture.

As a concrete example, let us consider Figure 18 where we analyze the failure modes of the self-trained model for two traffic sign classes – priority road ◈ and maximum speed 100 km/h (100). Here, each column corresponds to either an individual sample or a cluster or samples with similar properties (x-axis), the rows are robustness properties (y-axis), and a square (e.g., 🟦) denotes that a given sample is non-robust. We can see that for the priority road class, the translation property is the strongest and breaks the majority of the samples. For the maximum speed class, the properties are complementary to each other. More importantly, by clustering similar non-robust samples and comparing them to the robust ones, we discover important failure modes shown in Figure 18 (bottom). For ◈, the failure modes include semantic properties such as background being similar to the traffic sign, or strong direct sunlight. For (100), the failure modes are variations of extreme brightness conditions where both the background and traffic sign can be either very bright or very dark. Further, note that this analysis excluded most of the stronger properties defined in Section 4.2.1 for which the model robustness was very low.

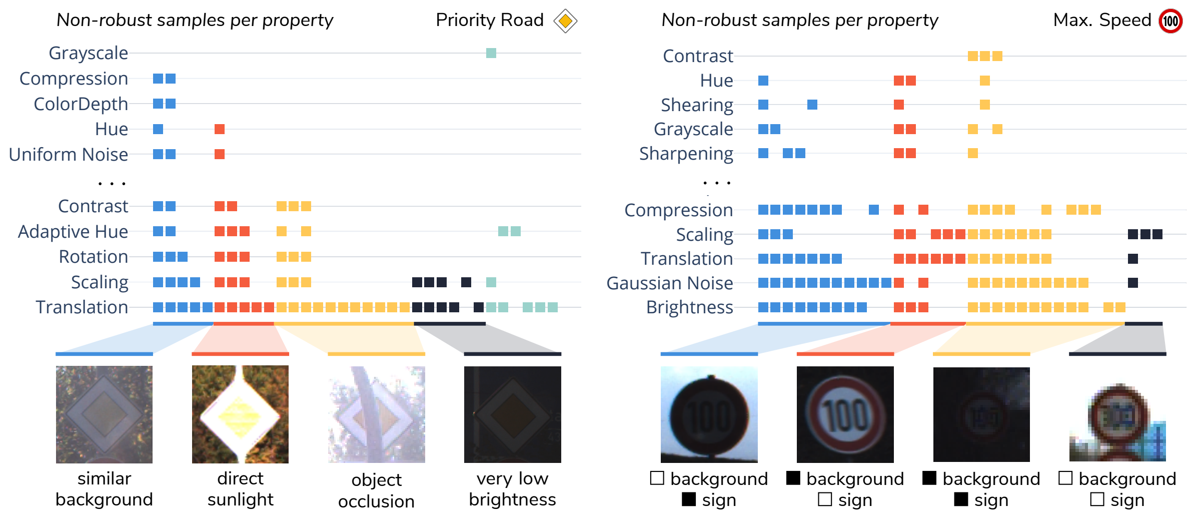


Figure 18: Example of failure mode analysis for the self-trained model. For a given property, each square denotes a non-robust sample (or a set of samples). Analysis of the robust (not shown) and non-robust samples identifies failure modes such as direct sunlight or various combinations of background and traffic sign brightness.

4.6 Enhancing datasets based on failure modes

Dataset | **Fixed** → **Failure Modes** → **Synthetic**

Obtaining high-quality datasets is a significant difficulty that most deep learning systems face. As a result, there has been an enormous interest in developing automatic data augmentation techniques [25, 21, 26, 16] as well as simulators explicitly tailored to self-driving cars [2, 1, 7, 5]. So far, we demonstrated how robustness properties are used to define a larger synthetic dataset on which the model robustness is tested. In this section, we discuss two extensions of this approach – leveraging failure modes to generate synthetic datasets and checking the model generalization on a new dataset.

Leverage failure modes to generate synthetic datasets In Section 4.5, we illustrated how failure modes can be used to gain insights into where the model is non-robust. Here, we discuss how they can be used to generate synthetic datasets, beyond those defined by the individual robustness properties. Concretely, let us use the maximum speed traffic sign , for which we identified four failure modes (shown in Figure 18). The failure modes correspond to different brightness conditions of the traffic sign and the image background. Our goal is to define new transformations to capture these failure modes, that is, transformations between bright and dark traffic signs and between bright and dark backgrounds. If these can be defined, then we can generate all four combinations synthetically.

To achieve this, several different approaches can be used. The first approach is to extend the range of allowed perturbations based on the failure model. For example, brightness can change both by a small amount locally (as done so far) but also by a large amount, as specified by the failure mode. This would allow more expressive changes without degrading the model performance by allowing unrestricted perturbations. The disadvantage of this approach is that the properties relevant to the failure mode, which can be complex, must be already supported. In the second approach, instead of transforming one property into another (i.e., bright to dark), we find a traffic sign with the desired property and insert it in the original image. To insert the image, one needs to: (i) compute a mask that denotes the location of the traffic sign, and (ii) apply a sequence of geometric and spatial transformations that overlays the inserted traffic sign over the existing one, as shown in Figure 19. A third possible approach is to train a generative adversarial model [19] that learns the transformation between samples with and without the desired property. However, while extremely useful, we do note that investigating these approaches in more detail is out of the scope of this report.

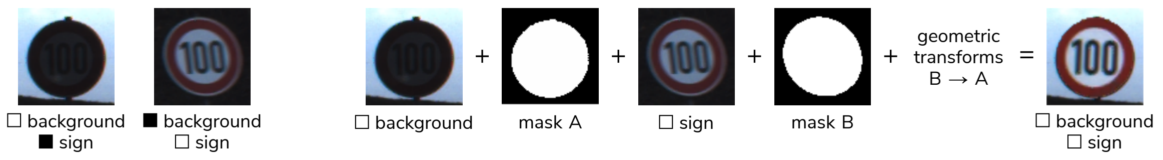


Figure 19: Failure modes examples for maximum speed traffic sign (left) and a transformation that converts between (right). To achieve this, the transformations merges two images with the desired properties (background and traffic sign is bright).

Generalization to a new dataset To evaluate generalization to a new dataset, we use the DFG traffic sign dataset [28], which consists of 200 traffic sign categories collected in Slovenia. We manually created a mapping to the classes represented in the GTSRB dataset and removed the remaining ones (we provide full details of our preprocessing in [13], Appendix E). The robustness of the best model (pre-trained) on this new dataset decreased by 0.9%-3.4% for all but one property, as summarized below:

BRIGHTNESS	CONTRAST	HUE	SHARPEN	BLUR	ROTATION	SHEAR	TRANSLATION	SCALE
-1.2%	**-1.1%**	**-2.6%**	**-2.5%**	**+1.7%**	**-3.4%**	**-0.9%**	**-2.3%**	**-3.4%**

The robustness did not decrease for blur because the DFG dataset has a significantly higher resolution. Further, the accuracy on the DFG dataset is significantly lower than that on the GTSRB dataset: it is 7.4% lower for the pre-trained model (99% to 91.6%) and 13.1% lower for the self-trained model (97.4% to 84.3%).

5 Conclusion

In this report, we presented a case study on assessing the reliability of two neural traffic sign classifiers. To conduct the assessment, we used the LatticeFlow platform to systematically test the neural networks using the test dataset and a set of robustness properties, which capture important environmental conditions that occur in practice. Concretely, the presented case study comprised the following tasks:

1. Eliciting strong robustness properties and important task-specific properties tailored to a specific domain (i.e., traffic signs in our case);
2. Testing using individual and composite robustness properties;
3. Finding common failure modes based on the results, including transformations that cause failures and test inputs where the model is brittle;

Our results show that the provided neural networks are robust to some of the basic generic properties. However, both networks lack robustness for stronger properties and important task-specific properties tailored to traffic signs (such as placing stickers on the road sign). In general, we observe that robustness properties with higher complexities lead to lower model robustness. Intuitively, this is because more complex properties define a larger space of image transformations, any of which can cause the model to fail. This is also partially because we tested the models for the worst-case image transformations. While this is necessary for correctly assessing where the model is failing and where it must be improved, it does provide a lower-bound on the robustness to average-case perturbations occurring benignly in the real-world.

Overall, the case study results give valuable insights into a systematic and general methodology for the reliability assessment of neural networks. Further, we demonstrate that the test results are useful for identifying important failure modes, which help enhance the dataset with challenging scenarios.

Finally, we note that we analyzed neural networks that, even though they achieve 99% accuracy, were trained by the BSI team using limited datasets and resources. The results hence do not transfer quantitatively to production models deployed in the current generation of self-driving cars.

References

[1] AI.Reverie. https://aireverie.com/, 2020.

[2] Anyverse. https://anyverse.ai/, 2020.

[3] Argoai. https://www.argo.ai/, 2020.

[4] balzano. https://www.balzano.ch/, 2020.

[5] cognata. https://www.cognata.com/traffic-signs/, 2020.

[6] cognex. https://connect.cognex.com/machine-vision-guide_LP_EN, 2020.

[7] CVEDIA. https://www.cvedia.com/syncity/, 2020.

[8] Deep learning of railway track faults using gpus. https://on-demand.gputechconf.com/gtc/2018/presentation/s8944-deep-learning-of-railway-track-faults-using-gpus.pdf, 2020.

[9] kheiron. https://www.kheironmed.com/, 2020.

[10] overjet. https://www.overjet.ai/, 2020.

[11] Tesla autopilot. https://www.tesla.com/autopilotAI, 2020.

[12] Waymo. https://waymo.com/, 2020.

[13] Pavol Bielik, Petar Tsankov, Andreas Krause, and Martin Vechev. Reliability assessment of traffic sign classifiers - extended version. Technical report, 2020. Available upon request from the Federal Office for Information Security.

[14] Alan C. Bovik. *Handbook of Image and Video Processing (Communications, Networking and Multimedia)*. Academic Press, Inc., USA, 2005.

[15] Philippe Cattin. Image restoration. https://miac.unibas.ch/SIP/06-Restoration.html. (accessed: 04.07.2020).

[16] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE International Conference on Computer Vision*, 2009.

[18] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust Physical-World Attacks on Deep Learning Visual Classification. In *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, 2018.

[19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[20] Joseph W Goodman. Statistical optics. *New York, Wiley-Interscience, 1985, 567 p.*, 1, 1985.

[21] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2731–2741, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[22] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, 2013.

[23] D. Kuan, A. Sawchuk, T. Strand, and P. Chavel. Adaptive restoration of images with speckle. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):373–383, 1987.

[24] Cassidy Laidlaw and Soheil Feizi. Functional adversarial attacks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 10408–10418, 2019.

[25] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 6665–6675. Curran Associates, Inc., 2019.

[26] Alexander J Ratner, Henry Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. Learning to compose domain-specific transformations for data augmentation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3236–3246. Curran Associates, Inc., 2017.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[28] Domen Tabernik and Danijel Skočaj. Deep Learning for Large-Scale Traffic-Sign Detection and Recognition. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

A Appendix

We provide full results and additional experiments, includng the formal specification of each robustness property, in the extended version of this report [13].