Verification of Neural Networks Against Convolutional Perturbations via Parameterised Kernels

Benedikt Brückner[1,2], **Alessio Lomuscio**[1,2]
[1]Safe Intelligence
[2]Imperial College London
benedikt@safeintelligence.ai, alessio@safeintelligence.ai

Abstract

We develop a method for the efficient verification of neural networks against convolutional perturbations such as blurring or sharpening. To define input perturbations, we use well-known camera shake, box blur and sharpen kernels. We linearly parameterise these kernels in a way that allows for a variation of the perturbation strength while preserving desired kernel properties. To facilitate their use in neural network verification, we develop an efficient way of convolving a given input with the parameterised kernels. The result of this convolution can be used to encode the perturbation in a verification setting by prepending a linear layer to a given network. This leads to tight bounds and a high effectiveness in the resulting verification step. We add further precision by employing input splitting as a branching strategy. We demonstrate that we are able to verify robustness on a number of standard benchmarks where the baseline is unable to provide any safety certificates. To the best of our knowledge, this is the first solution for verifying robustness against specific convolutional perturbations such as camera shake.

1 Introduction

As neural networks are increasingly deployed in safety-critical domains such as autonomous vehicles, aviation, or robotics, concerns about their reliability are rising. Networks have been shown to be vulnerable to *Adversarial Attacks*, i.e., perturbations that are often imperceptible, but change the output of a model for a given instance (Madry et al. 2017). Such adversarial examples have been shown to also exist in the physical world and pose a threat to algorithms deployed in practical applications (Tu et al. 2020). Neural Network Verification has been put forward as a way to address these issues by formally establishing that for a given input, a network is robust with respect to a set of specified perturbations; this property is often referred to as *local robustness* (Katz et al. 2017; Gehr et al. 2018; Singh et al. 2018).

Verification algorithms are usually divided into complete and incomplete approaches. Given enough time, complete methods are guaranteed to provide a definitive answer to the verification problem. In contrast, incomplete methods may not be able to answer the verification problem, returning an *undecided* result. Complete approaches often employ an exact encoding of the network at hand. They rely on techniques such as Mixed Integer Linear Programming (MILP) (Tjeng, Xiao, and Tedrake 2019; Anderson et al. 2020; Bunel et al. 2020) or Satisfiability Modulo Theories (SMT) (Pulina and Tacchella 2012; Katz et al. 2017). Incomplete verifiers employ methods such as Semidefinite Programming (Raghunathan, Steinhardt, and Liang 2018; Batten et al. 2021; Lan, Brueckner, and Lomuscio 2023), or bound propagation (Wang et al. 2018a,b; Singh et al. 2019b; Xu et al. 2021; Wang et al. 2021). They usually overapproximate the true behaviour of the neural network but can be made complete by combining them with a Branch and Bound (BaB) strategy. Stronger verifiers either employ tighter relaxations such as SDP-based ones or linear constraints that reason over multiple neurons simultaneously (Singh et al. 2019a; Müller et al. 2022; Ferrari et al. 2022; Zhang et al. 2022). State-of-the-art (SoA) verifiers achieve low runtimes through exploiting GPU-enabled parallelism (Brix et al. 2023b,a).

Early approaches established local robustness against norm-based perturbations, often referred to as *white noise* (Pulina and Tacchella 2012; Singh et al. 2018; Katz et al. 2019). A number of other perturbations were later proposed. Robustness to photometric perturbations such as brightness, contrast, hue or saturation changes as well as more expressive *bias field* perturbations can be verified by prepending suitable layers to a neural network (Kouvaros and Lomuscio 2018; Henriksen et al. 2021; Mohapatra et al. 2020). Verifiers were also extended to handle complex geometric perturbations such as rotations, translations, shearing or scaling, although the efficient verification against such perturbations requires further modifications and extensions (Kouvaros and Lomuscio 2018; Singh et al. 2019b; Balunovic et al. 2019; Batten et al. 2024). Other approaches focus on the efficient verification of robustness to occlusions (Mohapatra et al. 2020; Guo et al. 2023) or semantically rich perturbations in the latent space of generative models (Mirman et al. 2021; Hanspal and Lomuscio 2023).

More relevant to this work are previous investigations of camera shake effects. Guo et al. (2020) examine the performance of networks in the presence of motion blur. The results show a significant degradation of the performance of the models. This phenomenon can be modeled by applying a convolution operation using suitable kernels to a given in-

put image (Sun et al. 2015; Mei et al. 2019). Using different kernels, convolution operations can similarly be used to implement other image transformations which include box blur (Shapiro and Stockman 2001, pp.153–154) and sharpen (Arvo 1991, pp.50–56). Since many semantically interesting and realistic perturbations can be modelled by using convolutions, being able to verify robustness to perturbations in a kernel space is highly valuable. Some attempts at verifying the robustness of models to such perturbations have been made before. Paterson et al. (2021) encode contrast, haze and blur perturbations but only perform verification for haze while resorting to empirical testing for contrast and blur. One previous approach presents a general method which, if successful, certifies robustness of a network to all possible perturbations represented by a kernel of the given size (Mziou-Sallami and Adjed 2022). However, this generality comes at a cost. It leads to loose bounds and a high dimensionality of the perturbation which makes verification difficult, even more so for large networks. The universality also implies that counterexamples which are misclassified by the network may be difficult to interpret.

In this work we propose a new method which aims at the tight verification of networks against convolutional perturbations with a semantic meaning. Our key contributions are the following:

- We present an efficient, symbolic encoding of the perturbations and show how an arbitrary but constant input can efficiently be convolved with a linearly parameterised kernel using standard convolution operations.
- We present parameterised kernels for motion blur perturbations with various blurring angles as well as box blur and sharpen.
- Using standard benchmarks from past editions of the Verification of Neural Networks Competition (Brix et al. 2023b,a) as well as self-trained models, we show experimentally that verification is significantly easier with our method due to the tighter bounds and the low dimensionality of the perturbation. Our ablation study demonstrates that the existing method is unable to verify any properties on the networks we use. Our method certifies a majority of the properties for small kernel sizes and perturbation strengths while still being able to certify robustness in a number of cases for large kernel sizes and strengths.

2 Background

The notation used for the remainder of the paper is as follows: we use bold lower case letters $\boldsymbol{a}$ to denote vectors with $\boldsymbol{a}[i]$ representing the i-th element of a vector, bold upper case letters $\boldsymbol{A}$ to denote matrices with $\boldsymbol{A}[i,j]$ to denote the element in the i-th row and j-th column of a matrix and $\|\boldsymbol{a}\|_\infty := \sup_i |\boldsymbol{x}[i]|$ for the l_∞ norm of a vector. Since we focus on neural networks for image processing, we assume that the input of a given network will be an image and therefore refer to single entries in an input matrix as pixels.

2.1 Feed-forward Neural Networks

A feed-forward neural network (FFNN) is a function $f(\boldsymbol{x}) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ which is defined using the concatenation of $L \in \mathbb{N}$ *layers*. Each layer itself implements a function f_i and it holds that $f(\boldsymbol{x}) = f_L(f_{L-1}(\dots(f_1(\boldsymbol{x}))))$. Given an input $\boldsymbol{x}_0$ the output of a layer $1 \leq i \leq L$ is calculated in a recursive manner by applying the layer's operations to the output of the previous layer, i.e. $\boldsymbol{x}_i = f_i(\boldsymbol{x}_{i-1})$. The operation encoded by the i-th layer is $f_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ where n_i is the number of *neurons* in that layer. We assume each layer operation f_i to consist of two components: firstly, the application of a linear map $a_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}, \boldsymbol{x}_{i-1} \mapsto \boldsymbol{W}_i\boldsymbol{x}_{i-1} + \boldsymbol{b}_i$ for a weight matrix $\boldsymbol{W}_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and a bias vector $\boldsymbol{b}_i \in \mathbb{R}^{n_i}$ which yields the pre-activation vector $\hat{\boldsymbol{x}}_i$. And secondly the element-wise application of an *activation function* $\sigma_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ yielding the post-activation vector $\boldsymbol{x}_i$. In the verification literature networks are often assumed to use the piece-wise ReLU function $\mathrm{ReLU}(x) = \max(0, x)$, but verification is equally possible for other activations like sigmoid or tanh functions (Shi et al. 2024). The last layer of a given network normally does not include an activation function, σ_L would therefore be the identity map. In this work we focus on networks performing image classification where the input $\boldsymbol{x}_0$ is an image that needs to be categorised as belonging to one out of c classes. The final layer outputs $n_L = c$ classification scores and the predicted class for an image is $j = \arg\max_i x_L[i]$.

2.2 Neural Network Verification

Given a trained network f, the verification problem consists of formally showing that the output of the network is always contained in a linearly definable set $\mathcal{O} \subset \mathbb{R}^{n_L}$ for all inputs in a linearly definable input set $\mathcal{I} \subset \mathbb{R}^{n_0}$. Formally we aim to show that

$$\forall \boldsymbol{x}_0 \in \mathcal{I} : f(\boldsymbol{x}_0) \in \mathcal{O}$$

Inspired by adversarial attack paradigms, most works study the local robustness of networks to white noise contrained by the l_∞ norm (Bastani et al. 2016; Singh et al. 2018). Given an input $\bar{\boldsymbol{x}} \in \mathbb{R}^{n_0}$ which the network correctly classifies as belonging to class j', they do so by defining the input and output sets as

$$\mathcal{I} = \{\boldsymbol{x} \in \mathbb{R}^{n_0} \mid \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_\infty \leq \epsilon\}$$
$$\mathcal{O} = \{\boldsymbol{y} \in \mathbb{R}^{n_L} \mid \boldsymbol{y}[j'] > \boldsymbol{y}[j] \forall j \neq j'\}$$

where ϵ is the perturbation size for which the verification query should be solved.

SoA verifiers often employ bound propagation of some kind (Liu et al. 2019; Meng et al. 2022). If the bounds obtained at the final layer are tight enough, they can be used to answer the verification problem. The key difficulty in these approaches lies in the nonlinearity of the network activation functions. Convex relaxations of the functions are usually employed, but they induce an overapproximation error which can become significant for larger networks (Liu et al. 2019). Most methods employ a BaB mechanism which allows for a refinement of the network encoding if the problem cannot be solved with the initial encoding due to the relaxations being too coarse (De Palma et al. 2021; Ferrari et al. 2022; Shi et al. 2024). One branching strategy is *input splitting* which partitions the input space into subspaces

and has been found to be particularly effective for networks with low input dimensions (Wang et al. 2018b; Botoeva et al. 2020). *Neuron splitting* is also used for networks with high-dimensional perturbations where input splitting is less effective (Botoeva et al. 2020). This strategy splits the input space of a single neuron in the network into subspaces to allow for a more precise encoding of the activation functions. In the simple case of piece-wise ReLU activation functions, this can be done by splitting the function into its two linear pieces (Ferrari et al. 2022). Verification for high-dimensional perturbations such as norm-based ones is normally more challenging than verification for low-dimensional properties like brightness or contrast (Wang et al. 2018a). This is due to the *dimensionality* of the perturbations and the BaB strategies mentioned before. The typical return values of the verifiers consist of either *safe* (the network is robust under the given perturbation), *undecided* (the verifier could neither verify nor falsify the query, for example due to overly coarse relaxations), or *unsafe* (a concrete counterexample for which the network returns an incorrect result was found in the space of allowed perturbations).

2.3 Convolution

Convolution is a mathematical operation which is frequently used for processing inputs in signal processing. Most relevant for us is the discrete convolution operation on two-dimensional inputs, in our case images. For a given input matrix, it computes each element in the output matrix by multiplying the corresponding input value and its neighbours with different weights and then summing over the results. Given a two-dimensional input matrix $\boldsymbol{I}$ and a *kernel* matrix $\boldsymbol{K}$ we define the convolution of $\boldsymbol{I}$ with $\boldsymbol{K}$, often written as $\boldsymbol{I} * \boldsymbol{K}$, as follows (Goodfellow, Bengio, and Courville 2016, pp.331–334):

$$\boldsymbol{I} * \boldsymbol{K}[i, j] = \sum_{k,l} \boldsymbol{I}[i + k, j + l]\boldsymbol{K}[k, l]$$

Here (i, j) is a tuple of valid indices for the result, the output shape of $\boldsymbol{I} * \boldsymbol{K}$ can be computed based on a number of parameters (Dumoulin and Visin 2018). Signal processing literature often assumes that the kernel is flipped for convolution, but in line with common machine learning frameworks we omit this and refer to the above operation as convolution. The elements of $\boldsymbol{K}$ are usually normalised to sum to 1, for example if $\boldsymbol{I}$ is an image, since this preserves the brightness of the image. The output of a convolution is usually of a smaller size than the input, if an output of identical size is required, *padding* can be added to the image before the convolution. In early image processing algorithms linear filtering methods such as convolutions were frequently used for purposes such as edge detection or denoising with kernels being carefully designed by experts (Szeliski 2022, pp.100–101). Convolutional neural networks learn these kernels from data in order to perform a variety of tasks such as classification or object detection. When using suitable kernels, convolution can be used to apply effects such as sharpening to an image [Mei et al. 2019, Szeliski 2022, p.101]. We focus on the verification against camera shake or motion blur while also considering box blur and sharpen to demonstrate the generalisation of our method to other perturbations. Basic kernels for box blur (Shapiro and Stockman 2001, pp.153–154), sharpen (Arvo 1991, pp.50–56) and camera shake (Sun et al. 2015; Mei et al. 2019) are given in Figure 1 together with the identity kernel. The effect each of those kernels has on an image is also shown in the figure. Note that for inputs with multiple colour channels the convolution is performed independently for each channel.

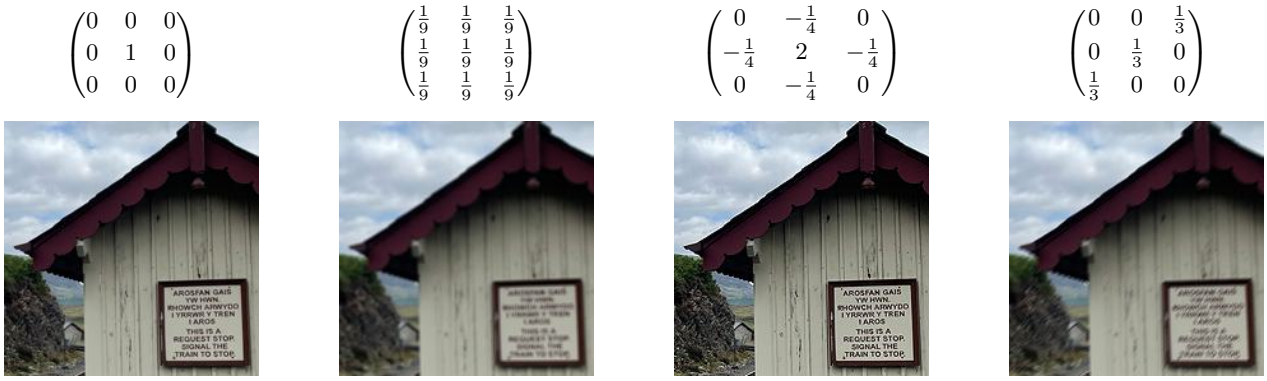
2.4 Convolutional Perturbations

Mziou-Sallami and Adjed (2022) propose an algorithm for verifying the robustness of a network to convolutional perturbations. They define the neighbourhood of a pixel in the input space for a given kernel size k, input image $\boldsymbol{I} \in \mathbb{R}^{d_1 \times d_2}$ and a pixel location tuple (i, j) as the set of all pixels inside a box of size $k \times k$ centered at the position (i, j). Fields of the box that lie outside the bounds of the input image are disregarded. A lower bound l and upper bound u for each pixel is calculated as the minimum and maximum element in that neighbourhood, respectively. These bounds are tight in the sense that they are attainable: the lower bound for a pixel can be realised through a kernel which has a 1 entry at the location of the minimum in the pixel's neighbourhood and zero elsewhere. A similar construction is possible for the maximum. If this operation is repeated for each pixel in the input image, one obtains two matrices $\boldsymbol{L} \in \mathbb{R}^{d_1 \times d_2}$ and $\boldsymbol{U} \in \mathbb{R}^{d_1 \times d_2}$ with lower and upper bounds for each pixel. A standard verifier can be used to certify robustness for a network on the given input by treating the perturbation as an l_∞, assuming that each pixel can vary independently between its lower and upper bound.

If verification is successful, the network is certified to be robust to any convolutional perturbation for which the values of the kernel lie in the $[0, 1]$ interval. However, since the robustness specification is very general, the bounds which are obtained from the method can be extremely loose. The assumption that variations of pixels are not coupled also means that the perturbation is high dimensional, leading to long runtimes and even looser bounds in layers deeper in the network. The combination of these factors means that even for the smallest kernel size of 3, Mziou-Sallami and Adjed (2022) obtain a verified accuracy of 30% in the best case and 0% in the worst case for small classifiers trained on the MNIST and CIFAR10 datasets. The approach is therefore unlikely to scale to larger networks.

3 Method

Phenomena such as motion blur cannot be modeled using standard techniques such as l_∞ perturbations since the computation of each output pixel's value is based not only on its original value, but also on the values of its neighbouring pixels. We parameterise specific kernels to model perturbations using convolution, allowing for the certification of robustness to specific types of convolutional perturbations while yielding tighter bounds. A major advantage of our approach is its simplicity. It can be implemented using standard oper-



(a) Identity (b) Box Blur (c) Sharpen (d) Motion Blur, $\phi = 45°$

Figure 1: Visualisation of the basic kernels used in this work

ations from a machine learning library to calculate the parameters of a linear layer prepended to the network to be verified. No special algorithms as in the case of geometric perturbations are required to perform efficient verification (Balunovic et al. 2019).

3.1 Convolution with Parameterised Kernels

In our work we use linearly parameterised kernels in the convolution operation. We refer to a kernel as linearly parameterised if each entry in the kernel matrix is an affine expression depending on a number of m variables. When convolving a constant input with such a kernel, the result is again a linear expression because of the linearity of the convolution operation.

Theorem 1. *Assume we are given an input image $\boldsymbol{I}$ and a parameterised kernel $\boldsymbol{K}$ defined as*

$$\boldsymbol{K} = \sum_{i=1}^{m} \boldsymbol{A}_i \cdot z_i + \boldsymbol{B}$$

where $z_i \in \mathbb{R}$. $\boldsymbol{A}_i$ and $\boldsymbol{B}$ are a number of coefficient matrices and a bias matrix, respectively, which have the same shape as $\boldsymbol{K}$. Then we have:

$$\boldsymbol{I} * \boldsymbol{K} = \sum_{i=1}^{m} (\boldsymbol{I} * \boldsymbol{A}_i)\, z_i + \boldsymbol{I} * \boldsymbol{B} \tag{1}$$

This theorem allows us to compute the result of a convolution with a parameterised kernel by separately convolving the input with each coefficient matrix and the bias matrix (Equation 1). The z_i variables can be ignored during these computations which means that the convolution operations need to only be executed on $(m+1)$ constant matrices. Standard convolution implementations from a machine learning library can thus be used for this computation.

3.2 Parameterised Kernels for Modelling Camera Shake

To enable the verification of a network to a range of perturbation strengths, we model a linear transition from the identity kernel to a desired perturbation kernel $\boldsymbol{P}$ using a variable $z \in [0, 1]$. Two initial conditions are given for the kernel: it should be equal to the identity kernel for $z = 0$, and equal to the desired perturbation kernel such as those in Figure 1 for $z = 1$. An affine function is unambiguously defined by these two points (that it intercepts), and we can compute the slope and intercept for each entry in the kernel. Example 1 shows this derivation for the 3×3 motion blur kernel with a blur angle of $\phi = 45°$ introduced in Section 2.3.

Example 1. *The initial conditions for our parameterisation are:*

$$\boldsymbol{P}_{z=0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \boldsymbol{P}_{z=1} = \begin{pmatrix} 0 & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & 0 \\ \frac{1}{3} & 0 & 0 \end{pmatrix}$$

We assume that each kernel entry is of the form $p(z) = az+b$ where $z \in \mathbb{R}$ is a variable and $a, b \in \mathbb{R}$ are parameters. Since we have two unknowns a, b and two points that the function passes through from the initial conditions, we can solve for a, b to obtain our parameters for each kernel entry. In the camera shake case for $\phi = 45°$ we have three types of entries: The center entry, the non-center entries on the antidiagonal running from the top right to the bottom left of the matrix, and the entries that do not lie on the antidiagonal.

Parameterisation for the Center Entry. *For the center entry we have $p(0) = 1$ and $p(1) = \frac{1}{3}$. This results in the constraints*

$$p(0) = 1 = a \cdot 0 + b \tag{2}$$

$$p(1) = \frac{1}{3} = a \cdot 1 + b \tag{3}$$

Solving for a, b yields $a = -\frac{2}{3}, b = 1$ and therefore $p(z) = -\frac{2}{3}z + 1$.

Parameterisation for the Antidiagonal Entries. *For non-center entries on the antidiagonal the initial conditions are*

$$p(0) = 0 = a \cdot 0 + b \tag{4}$$

$$p(1) = \frac{1}{3} = a \cdot 1 + b \tag{5}$$

Equation 4 implies that we have $b = 0$ and then $a = \frac{1}{3}$ follows from Equation 5. We obtain $p(z) = \frac{1}{3}z$

Parameterisation for the Off-Antidiagonal Entries. *For the entries off the antidiagonal we have $p(0) = p(1) = 0$ and therefore $p(z) = 0$.*

In conclusion, we get $\boldsymbol{P} = \boldsymbol{A} \cdot z + \boldsymbol{B}$ with

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & \frac{1}{3} \\ 0 & -\frac{2}{3} & 0 \\ \frac{1}{3} & 0 & 0 \end{pmatrix}, \boldsymbol{B} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

In a similar way to the above, parameterisations can be derived for other kernel sizes, different motion blur angles and other perturbations such as blur or sharpen as shown in Figure 1. We do not consider kernel sizes that are even numbers in our experiments since the identity kernel is not well-defined for even kernel sizes. An identity kernel can be approximated by defining the four entries around the true center of the kernel to have a value of $\frac{1}{2}$ with all other values being zero. However, this kernel may still add a noticeable blur to the image.

3.3 Integration of Convolutional Perturbations into Neural Network Verifiers

Given the parameterised kernels from Section 3.2, we can use Theorem 1 to devise a method for easily verifying the robustness of neural networks to the perturbations the kernels encode. Since the parameterised kernels we introduce only depend on a single variable we omit the indexing of the variable z and the coefficient matrix $\boldsymbol{A}$. We borrow the popular idea of using additional layers that encode a perturbation which are then prepended to a network for verification (Kouvaros and Lomuscio 2018; Mohapatra et al. 2020; Guo et al. 2023). Assume a trained neural network f is given together with a correctly classified input image in vectorised form $\bar{\boldsymbol{x}} \in \mathbb{R}^{o_c \cdot o_h \cdot o_w}$ where o_c, o_h, o_w are the image's number of channels, height and width, respectively. We first reshape the vector $\bar{\boldsymbol{x}}$ into the original shape of the image (o_c, o_h, o_w) to obtain an input tensor $\boldsymbol{I}$. This step is necessary for the convolution operation to be applicable. We then separately convolve $\boldsymbol{I}$ with $\boldsymbol{A}$ and $\boldsymbol{B}$ to obtain $\boldsymbol{R_A} := (\boldsymbol{I} * \boldsymbol{A})$ and $\boldsymbol{R_B} := (\boldsymbol{I} * \boldsymbol{B})$. For inputs with multiple channels each channel is convolved independently with the same kernels $\boldsymbol{A}, \boldsymbol{B}$ so the output of the convolution has the same number of channels as the input.

To encode the perturbation in a network layer, we reshape the resulting matrices of these convolutions to be vectors $\boldsymbol{r_A}, \boldsymbol{r_B} \in \mathbb{R}^{o_c \cdot o_h \cdot o_w}$ again. We then prepend a matrix multiplication layer to the network which computes $\tilde{\boldsymbol{A}} \cdot z + \tilde{\boldsymbol{B}}$ where $\tilde{\boldsymbol{A}} = \boldsymbol{r_A}, \tilde{\boldsymbol{B}} = \boldsymbol{r_B}$ are parameters for the layer that are set for each verification query and $z \in \mathbb{R}$ is the input to the network controlling the strength of the perturbation. Similarly to the existing approaches which encode properties to verify in a layer prepended to the network, the image information is now encoded in the parameters of this new layer. Despite the fact that all parameterisations used in this work only depend on one variable, the method generalises to the case where the number of coefficient matrices in the parameterisation m is greater than one. In those cases we perform $m + 1$ separate convolutions and obtain m matrices $\boldsymbol{R_{A_1}}, \ldots, \boldsymbol{R_{A_m}}$ and one matrix $\boldsymbol{R_B}$ which are reshaped into vectors $\boldsymbol{r_{A_1}}, \ldots, \boldsymbol{r_{A_m}}, \boldsymbol{r_B}$. Assuming $\boldsymbol{r_{A_i}} \in \mathbb{R}^{o_c \cdot o_h \cdot o_w}$ are column vectors, they can be concatenated horizontally to form a parameter matrix $\tilde{\boldsymbol{A}} \in \mathbb{R}^{o_c \cdot o_h \cdot o_w, m}$. The input to the network in this case would be a vector $z \in \mathbb{R}^m$ for parameterising the perturbation. The prepended layer then computes the matrix-vector product $\tilde{\boldsymbol{A}}z$, adds the bias $\boldsymbol{B}$ to it and feeds the resulting vector of size $o_c \cdot o_h \cdot o_w$ into the first layer of the original network. The robustness of the resulting network can be checked by using standard neural network verifiers. Since the input to the augmented network is low-dimensional, in our case even just one-dimensional, input splitting as a branching strategy is particularly effective for verification (Botoeva et al. 2020).

4 Evaluation

To evaluate the method described in the previous section, we used Venus (Kouvaros and Lomuscio 2021), a robustness verification toolkit that uses both Mixed-Integer Linear Programming and Symbolic Interval Propagation to solve verification problems. The toolkit was extended to process modified vnnlib files as used in the most recent Verification of Neural Networks Competition (VNNCOMP23) which encoded the perturbations (Brix et al. 2023a). Venus uses PyTorch (Paszke et al. 2019) for efficient vectorised computations. We enabled its SIP solver and its adversarial attacks engine, and used the default settings for anything else. Our method is implemented in PyTorch and reads a vnnlib file, builds the parameterised kernels as described in Section 3.2 and convolves the input with them using PyTorch's `Conv2d` operation. An additional layer encoding the perturbation was prepended to the network as described in Section 3.3, and verification with input splitting was run on the augmented network using Venus. The experiments were conducted on a Fedora 35 server equipped with an AMD EPYC 7453 28-Core Processor and 512GB of RAM.

The performance of robustness verification for the proposed perturbations was evaluated on three benchmarks from previous editions of the Verification of Neural Networks Competition (Müller et al. 2022). mnist_fc is a classification benchmark which consists of three different networks with 2, 4 and 6 layers with 256 ReLU nodes each that is trained on the MNIST dataset. Oval21 contains three convolutional networks trained on the CIFAR10 dataset. Two of them consist of two convolutional layers followed by two fully-connected layers while the third one has two additional convolutional layers, the number of network activations ranges from 3172 to 6756. Sri_resnet_a is a ReLU-based ResNet with one convolutional layer, three ResBlocks and two fully-connected layers trained using adversarial training on the CIFAR10 dataset. For our resnet18 benchmark, we trained a ResNet18 model from the TorchVision package (version 0.16.0) on the CIFAR10 dataset. The network has 11.7M parameters. For each VNNCOMP benchmark we took the properties from VNNCOMP and changed

		Box Blur		Sharpen		Motion Blur 0°		Motion Blur 45°		Motion Blur 90°		Motion Blur 135°	
s	strength	v/us/to	time	v/us/to	time	v/us/to	time	v/us/to	time	v/us/to	time	v/us/to	time
3	0.2	45/0/0	87	45/0/0	80	45/0/0	78	45/0/0	77	45/0/0	78	44/1/0	78
3	0.4	44/1/0	77	45/0/0	77	45/0/0	77	45/0/0	78	45/0/0	77	44/1/0	78
3	0.6	44/1/0	78	45/0/0	77	44/1/0	76	44/1/0	78	44/1/0	76	44/1/0	77
3	0.8	44/1/0	80	45/0/0	79	44/1/0	77	44/1/0	79	44/1/0	76	44/1/0	81
3	1.0	44/1/0	83	45/0/0	79	44/1/0	79	44/1/0	80	44/1/0	78	42/3/0	83
5	0.2	44/1/0	75	45/0/0	76	44/1/0	77	44/1/0	76	44/1/0	76	44/1/0	76
5	0.4	44/1/0	81	45/0/0	77	44/1/0	78	44/1/0	77	44/1/0	77	44/1/0	82
5	0.6	43/2/0	87	45/0/0	81	44/1/0	80	44/1/0	83	44/1/0	78	38/7/0	85
5	0.8	41/4/0	91	45/0/0	84	43/2/0	83	44/1/0	88	44/1/0	82	35/10/0	87
5	1.0	36/9/0	91	45/0/0	88	42/3/0	84	41/4/0	90	44/1/0	87	31/14/0	89
7	0.2	44/1/0	77	45/0/0	76	44/1/0	77	44/1/0	78	44/1/0	77	44/1/0	76
7	0.4	42/3/0	88	45/0/0	77	44/1/0	80	44/1/0	82	44/1/0	78	38/7/0	82
7	0.6	34/11/0	90	45/0/0	86	42/3/0	84	41/4/0	89	44/1/0	84	34/11/0	87
7	0.8	25/20/0	87	44/1/0	88	37/8/0	87	37/8/0	91	42/3/0	88	24/21/0	85
7	1.0	20/25/0	86	43/2/0	90	33/12/0	86	30/15/0	91	39/6/0	90	22/23/0	87
9	0.2	44/1/0	78	45/0/0	77	44/1/0	77	44/1/0	77	44/1/0	76	42/3/0	78
9	0.4	36/9/0	86	45/0/0	82	42/3/0	82	42/3/0	86	44/1/0	83	35/10/0	84
9	0.6	24/21/0	86	44/1/0	87	36/9/0	85	34/11/0	90	40/5/0	87	26/19/0	86
9	0.8	17/28/0	83	43/2/0	90	29/16/0	87	27/18/0	89	39/6/0	91	20/25/0	86
9	1.0	9/36/0	77	43/2/0	94	26/19/0	87	16/29/0	83	36/9/0	93	13/32/0	82

Table 1: Experimental evaluation on mnist_fc. Each query is run with a timeout of 1800 seconds. s is the filter size and v/us/to denote the number of verified/unsafe/timeout instances, respectively.

the perturbation type to one of ours before running the experiments. The timeout for each query was set to 1800 seconds. For resnet18 we selected 50 correctly classified instances from the CIFAR10 test set for verification.

For each of the benchmarks, we tested perturbations with kernel sizes of $3, 5, 7$ and 9. We varied the upper bound of the perturbation strength which we simply denote as *strength* in the following, the lower bound for the strength is zero. For example, a perturbation strength of 0.4 means that for the parameterised kernel, the variable z is allowed to vary within the interval $[0, 0.4]$. The results for mnist_fc are presented in Table 1, those for resnet18 in Table 2.

We found that our method scales well to very large networks such as ResNet18 and found that the kernel size and perturbation strength had a much larger impact on the verifiability of a model than its size. Verification on all benchmarks was fast due to the low dimensionality of our perturbations. Verification for small perturbation strengths was successful for nearly all instances, irrespective of the kernel size s. For small kernel sizes such as $s = 3$ we further observed that verification was successful even for very large strengths. For large kernel sizes and large perturbation strengths, unsafe cases were more likely to be found which is not surprising given that the degree of corruption for e.g. box blur with a kernel size of 9 and a perturbation strength of 1 is substantial. The differences in robustness to different types of perturbations were also noteworthy. While robustness deteriorated severely for box blur and camera shake when larger kernel sizes or perturbation strengths were considered, networks retained a high verified robustness against sharpen perturbations. This intuitively makes sense. While blurring often induces an information loss which can make it hard to restore the original information of the image, sharpening emphasises the image texture and can strengthen edges in the image. This robustness to large sharpen perturbation strengths could be observed for both MNIST and CIFAR10.

For mnist_fc we also observed that the robustness of the network to camera shake perturbations was highly dependent on the perturbation angle, especially for a kernel size of 9. While the networks were vulnerable to camera shake along the 45° and 135° axis, they were much more robust to blurring along the 0° axis. Motion blurring along the 90° axis affected the networks to the least degree with verified accuracies still being extremely high for strong perturbations, even for a kernel size of 9 and a perturbation strength of 1.0. The differences in robustness to different camera shake angles were also observable for resnet18, oval21 and sri_resnet_a, even though they were less prominent.

For baseline comparisons we reimplemented the method presented in (Mziou-Sallami and Adjed 2022) and the resulting verification queries were once again solved by Venus. Since the resulting perturbations are high-dimensional, we used activation splitting instead of input splitting as a better performing branching strategy. If the baseline method verifies robustness for a given model, input and kernel size, the model is robust to any perturbation that can be encoded with a kernel of the given size for that input. However, Table 3 shows that the baseline perturbations lead to loose bounds due to the high dimensionality of the perturbations combined with the already loose bounds for each pixel's value for large neighbourhoods. Even for a small kernel size of 3, we found that no properties could be verified for the net-

		Box Blur		Sharpen		Motion Blur 0°		Motion Blur 45°		Motion Blur 90°		Motion Blur 135°	
s	strength	v/us/to	time	v/us/to	time	v/us/to	time	v/us/to	time	v/us/to	time	v/us/to	time
3	0.2	46/4/0	4781	46/4/0	4235	47/3/0	3924	46/4/0	4985	46/4/0	3739	46/4/0	4950
3	0.4	45/5/0	7156	46/4/0	6241	47/3/0	6092	45/5/0	7286	46/4/0	5729	45/5/0	7252
3	0.6	43/7/0	8519	46/4/0	7677	45/5/0	7284	44/6/0	8709	45/5/0	6895	40/10/0	7943
3	0.8	37/13/0	8864	46/4/0	8621	44/6/0	8358	41/9/0	9904	43/7/0	7732	36/14/0	8398
3	1.0	32/18/0	9211	46/4/0	9325	42/8/0	8957	40/10/0	11442	42/8/0	8374	35/15/0	10120
5	0.2	46/4/0	6415	46/4/0	5454	47/3/0	5733	46/4/0	6291	46/4/0	5297	45/5/0	6190
5	0.4	42/8/0	8594	46/4/0	7614	42/8/0	7558	45/5/0	8976	45/5/0	7727	43/7/0	8582
5	0.6	35/15/0	9963	46/4/0	8942	40/10/0	8973	38/12/0	10372	43/7/0	8958	37/13/0	9935
5	0.8	24/26/0	9641	46/4/0	10255	33/17/0	9452	29/21/0	10706	36/14/0	9271	26/24/0	8919
5	1.0	22/28/0	11857	43/7/0	10658	30/20/0	10248	26/24/0	12164	34/16/0	10852	23/27/0	10269
7	0.2	46/4/0	7295	46/4/0	6155	44/6/0	6179	46/4/0	7022	46/4/0	6026	45/5/0	6846
7	0.4	42/8/0	9919	45/5/0	8310	41/9/0	8312	43/7/0	9798	45/5/0	8526	43/7/0	9722
7	0.6	38/12/0	13899	45/5/0	9647	35/15/0	9859	36/14/0	11507	40/10/0	10043	37/13/0	11704
7	0.8	22/28/0	11823	43/7/0	10715	28/22/0	10372	27/23/0	11890	32/18/0	10698	28/22/0	11971
7	1.0	14/36/0	10464	40/10/0	11178	24/26/0	11100	21/29/0	11747	21/29/0	8213	19/31/0	10387
9	0.2	46/4/0	7914	46/4/0	6454	45/5/0	6822	46/4/0	7596	46/4/0	6371	45/5/0	7507
9	0.4	43/7/0	11115	44/6/0	8482	41/9/0	9282	43/7/0	10660	45/5/0	9120	42/8/0	10301
9	0.6	37/13/0	14980	43/7/0	10028	34/16/0	11029	35/15/0	12874	40/10/0	11312	35/15/0	12814
9	0.8	22/28/0	14201	41/9/0	11066	23/27/0	10160	27/23/0	14035	25/25/0	9322	27/23/0	13803
9	1.0	10/40/0	7911	40/10/0	12457	19/31/0	10685	17/33/0	12090	17/33/0	8698	21/29/0	13471

Table 2: Experimental evaluation on resnet18. Each query was run with a timeout of 1800 seconds. s is the filter size and v/us/to denote the number of verified/unsafe/timeout instances, respectively.

	mnist_fc		resnet18		oval21		sri_resnet_a	
s	v/us/to	time	v/us/to	time	v/us/to	time	v/us/to	time
3	0/45/0	70	0/50/0	133	0/30/0	258	0/71/1	7311
5	0/45/0	70	0/50/0	133	0/30/0	49	0/72/0	113
7	0/44/1	7268	0/50/0	133	0/30/0	2298	0/71/1	7311
9	0/44/1	7271	0/50/0	129	0/30/0	48	0/72/0	112

Table 3: Ablation Study

works we consider. In these cases, bounds became so loose that it is easy for the verifier to find concrete counterexamples in the majority of cases. The generality of the perturbation specifications in the baseline is one of its strengths, but at the same time limits its practical use since we showed that it does not scale to larger models or input sizes. Our method is less general and we need to do a separate verification run for each kernel used for modelling a convolutional perturbation. However, our approach scales to much larger networks and therefore enables robustness certification in scenarios where the baseline fails.

In summary, our findings indicate that the method here presented enables the verification of networks against a range of perturbations that can be modeled through parameterised kernels. The method provides tight bounds, permitting the verification of much larger networks, and allows for an efficient verification due the low dimensionality of the perturbations.

5 Conclusions

Verification against camera shake and related convolutional perturbations is important since these phenomena are likely to appear in the real world. So far, verification against such perturbations was only possible by using an algorithm that yielded counterexamples which are difficult to interpret and produced loose bounds, preventing the robustness verification of larger networks or networks with large input sizes. The here proposed approach based on parameterised kernels is easy to implement and allows for the verification of network robustness to a number of semantically interesting perturbations. We demonstrated the effectiveness of the method on standard benchmarks from VNNComp and proved that it is able to verify properties for networks that cannot be solved by the existing baseline. As we showed, the method can be used to identify weaknesses of models to specific types of convolutional perturbations that might otherwise remain hidden such as motion blurring along axes of a specific angle. Since it is easy to design parameterised kernels for additional perturbation types besides those presented, we expect that more convolutional perturbations of practical interest will be developed in the future, contributing to more thorough robustness checks for deployed AI systems.

Acknowledgements

Benedikt Brückner acknowledges support from the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (EP/S023356/1]. Alessio Lomuscio acknowledges support from the Royal Academy of Engineering via a Chair of Emerging Technologies.

References

Anderson, R.; Huchette, J.; Ma, W.; Tjandraatmadja, C.; and Vielma, J. 2020. Strong mixed-integer programming formulations for trained neural networks. In *Integer Programming and Combinatorial Optimization*, volume 11480 of *LNCS*, 27–42. Springer.

Arvo, J. 1991. *Graphics Gems II*. Elsevier Science.

Balunovic, M.; Baader, M.; Singh, G.; Gehr, T.; and Vechev, M. 2019. Certifying Geometric Robustness of Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS19)*, 15313–15323. Curran Associates, Inc.

Bastani, O.; Ioannou, Y.; Lampropoulos, L.; Vytiniotis, D.; Nori, A.; and Criminisi, A. 2016. Measuring Neural Net Robustness with Constraints. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS16)*, 2613–2621.

Batten, B.; Kouvaros, P.; Lomuscio, A.; and Zheng, Y. 2021. Efficient Neural Network Verification via Layer-based Semidefinite Relaxations and Linear Cuts. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*, 2184–2190. ijcai.org.

Batten, B.; Zheng, Y.; De Palma, A.; Kouvaros, P.; and Lomuscio, A. 2024. Verification of Geometric Robustness of Neural Networks via Piecewise Linear Approximation and Lipschitz Optimisation. In *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI24)*, 2362–2369. IOS Press.

Botoeva, E.; Kouvaros, P.; Kronqvist, J.; Lomuscio, A.; and Misener, R. 2020. Efficient Verification of Neural Networks via Dependency Analysis. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI20)*, 3291–3299. AAAI Press.

Brix, C.; Bak, S.; Liu, C.; and Johnson, T. T. 2023a. The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results. *arXiv preprint arXiv:2312.16760*.

Brix, C.; Müller, M. N.; Bak, S.; Johnson, T. T.; and Liu, C. 2023b. First Three Years of the International Verification of Neural Networks Competition (VNN-COMP). *arXiv preprint arXiv:2301.05815*.

Bunel, R.; Lu, J.; Turkaslan, I.; Kohli, P.; Torr, P.; and Mudigonda, P. 2020. Branch and Bound for Piecewise Linear Neural Network Verification. *Journal of Machine Learning Research*, 21(42): 1–39.

De Palma, A.; Bunel, R.; Desmaison, A.; Dvijotham, K.; Kohli, P.; Torr, P.; and Kumar, M. P. 2021. Improved branch and bound for neural network verification via lagrangian decomposition. *arXiv preprint arXiv:2104.06718*.

Dumoulin, V.; and Visin, F. 2018. A guide to convolution arithmetic for deep learning. *arXiv preprint 1603.07285*.

Ferrari, C.; Mueller, M.; Jovanović, N.; and Vechev, M. 2022. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *Proceedings of the 10th International Conference on Learning Representations (ICLR22)*. Openreview.net.

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. 2018. AI^2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy (SP18)*, 3–18. IEEE.

Goodfellow, A.; Bengio, Y.; and Courville, A. 2016. *Deep learning*, volume 1. MIT press Cambridge.

Guo, Q.; Juefei-Xu, F.; Xie, X.; Ma, L.; Wang, J.; Yu, B.; Feng, W.; and Liu, Y. 2020. Watch out! Motion is Blurring the Vision of Your Deep Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS20)*, volume 33, 975–985. Curran Associates, Inc.

Guo, X.; Zhou, Z.; Zhang, Y.; Katz, G.; and Zhang, M. 2023. OccRob: Efficient SMT-Based Occlusion Robustness Verification of Deep Neural Networks. In *Proceedings of the 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS23)*, volume 13993 of *Lecture Notes in Computer Science*, 208–226. Springer.

Hanspal, H.; and Lomuscio, A. 2023. Efficient Verification of Neural Networks against LVM-based Specifications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR23)*. IEEE.

Henriksen, P.; Hammernik, K.; Rueckert, D.; and Lomuscio, A. 2021. Bias Field Robustness Verification of Large Neural Image Classifiers. In *Proceedings of the 32nd British Machine Vision Conference (BMVC21)*. BMVA Press.

Katz, G.; Barrett, C.; Dill, D.; Julian, K.; and Kochenderfer, M. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV17)*, volume 10426 of *Lecture Notes in Computer Science*, 97–117. Springer.

Katz, G.; Huang, D.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D.; Kochenderfer, M.; and Barrett, C. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV19)*, 443–452.

Kouvaros, P.; and Lomuscio, A. 2018. Formal Verification of CNN-based Perception Systems. *arXiv preprint arXiv:1811.11373*.

Kouvaros, P.; and Lomuscio, A. 2021. Towards Scalable Complete Verification of ReLU Neural Networks via Dependency-based Branching. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*, 2643–2650. ijcai.org.

Lan, J.; Brueckner, B.; and Lomuscio, A. 2023. A Semidefinite Relaxation Based Branch-and-Bound Method for Tight Neural Network Verification. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI23)*, 14946–14954. AAAI Press.

Liu, C.; Arnon, T.; Lazarus, C.; Barrett, C.; and Kochenderfer, M. 2019. Algorithms for verifying deep neural networks. *arXiv preprint 1903.06758*.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Mei, J.; Wu, Z.; Chen, X.; Qiao, Y.; Ding, H.; and Jiang, X. 2019. Deepdeblur: Text Image Recovery from Blur to Sharp. *Multimedia tools and applications*, 78: 18869–18885.

Meng, M.; Bai, G.; Teo, S.; Hou, Z.; Xiao, Y.; Lin, Y.; and Dong, J. 2022. Adversarial Robustness of Deep Neural Networks: A Survey from a Formal Verification Perspective. *IEEE Transactions on Dependable and Secure Computing*.

Mirman, M.; Hägele, A.; Bielik, P.; Gehr, T.; and Vechev, M. 2021. Robustness certification with generative models. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI21)*, 1141–1154. Association for Computing Machinery.

Mohapatra, J.; Weng, T.-W.; Chen, P.-Y.; Liu, S.; and Daniel, L. 2020. Towards Verifying Robustness of Neural Networks Against A Family of Semantic Perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR20)*, 241–249. IEEE.

Müller, M.; Brix, C.; Bak, S.; Liu, C.; and Johnson, T. 2022. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. *arXiv preprint arXiv:2212.10376*.

Müller, M. N.; Makarchuk, G.; Singh, G.; Püschel, M.; and Vechev, M. 2022. PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. *Proceedings of the ACM on Programming Languages*, 6.

Mziou-Sallami, M.; and Adjed, F. 2022. Towards a Certification of Deep Image Classifiers against Convolutional Attacks. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART22)*, 419–428.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS19)*, 8024–8035. Curran Associates, Inc.

Paterson, C.; Wu, H.; Grese, J.; Calinescu, R.; Păsăreanu, C.; and Barrett, C. 2021. DeepCert: Verification of Contextually Relevant Robustness for Neural Network Image Classifiers. In *Proceedings of the 24th International Conference on Computer Safety, Reliability, and Security (SAFECOMP21)*, Lecture Notes in Computer Science, 3–17. Springer.

Pulina, L.; and Tacchella, A. 2012. Challenging SMT solvers to verify neural networks. *AI Communications*, 25(2): 117–135.

Raghunathan, A.; Steinhardt, J.; and Liang, P. S. 2018. Semidefinite relaxations for certifying robustness to adversarial examples. In *Proceedings of the 32th International Conference on Neural Information Processing Systems (NIPS18)*, 10900–10910.

Shapiro, L.; and Stockman, G. 2001. *Computer Vision*. Prentice Hall.

Shi, Z.; Jin, Q.; Kolter, J.; Jana, S.; Hsieh, C.-J.; and Zhang, H. 2024. Formal Verification for Neural Networks with General Nonlinearities via Branch-and-Bound.

Singh, G.; Ganvir, R.; Püschel, M.; and Vechev, M. 2019a. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems*, 32.

Singh, G.; Gehr, T.; Mirman, M.; Püschel, M.; and Vechev, M. 2018. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems (NeurIPS18)*, 10802–10813.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019b. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL): 41.

Sun, J.; Cao, W.; Xu, Z.; and Ponce, J. 2015. Learning a Convolutional Neural Network for Non-Uniform Motion Blur Removal. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR15)*, 769–777.

Szeliski, R. 2022. *Computer Vision: Algorithms and Applications*. Springer.

Tjeng, V.; Xiao, K.; and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proceedings of the 7th International Conference on Learning Representations (ICLR19)*.

Tu, J.; Ren, M.; Manivasagam, S.; Liang, M.; Yang, B.; Du, R.; Cheng, F.; and Urtasun, R. 2020. Physically Realizable Adversarial Examples for LiDAR Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR20)*, 13713–13722.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018a. Efficient Formal Safety Analysis of Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS18)*, 6367–6377. Curran Associates, Inc.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018b. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proceedings of the 27th USENIX Security Symposium (USENIX18)*.

Wang, S.; Zhang, H.; Xu, K.; Lin, X.; Jana, S.; Hsieh, C.; and Kolter, J. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *arXiv preprint arXiv:2103.06624*.

Xu, K.; Zhang, H.; Wang, S.; Wang, Y.; Jana, S.; Lin, X.; and Hsieh, C.-J. 2021. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *Proceedings of the 9th International Conference on Learning Representations (ICLR21)*. OpenReview.net.

Zhang, H.; Wang, S.; Xu, K.; Li, L.; Li, B.; Jana, S.; Hsieh, C.-J.; and Kolter, J. Z. 2022. General Cutting Planes for Bound-Propagation-Based Neural Network Verification. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS22)*, 1656–1670. Curran Associates, Inc.