Safety Verification of Deep Neural Networks*

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, Min Wu

University of Oxford

Abstract. Deep neural networks have achieved impressive experimental results in image classification, but can surprisingly be unstable with respect to adversarial perturbations, that is, minimal changes to the input image that cause the network to misclassify it. With potential applications including perception modules and end-to-end controllers for self-driving cars, this raises concerns about their safety. We develop the first SMT-based automated verification framework for feed-forward multi-layer neural networks that works directly with the code of the network, exploring it layer by layer. We define safety for a region around a data point in a given layer by requiring that all points in the region are assigned the same class label. Working with a notion of a manipulation, a mapping between points that intuitively corresponds to a modification of an image, we employ discretisation to enable exhaustive search of the region. Our method can guarantee that adversarial examples are found for the given region and set of manipulations. If found, adversarial examples can be shown to human testers and/or used to fine-tune the network, and otherwise the network is declared safe for the given parameters. We implement the techniques using Z3 and evaluate them on state-of-the-art networks, including regularised and deep learning networks.

1 Introduction

Deep neural networks have achieved impressive experimental results in image classification, matching the cognitive ability of humans [17] in complex tasks with thousands of classes. Many applications are envisaged, including their use as perception modules and end-to-end controllers for self-driving cars [12]. Let $\mathbb{R}^n$ be a vector space of images (points) that we wish to classify and assume that $f : \mathbb{R}^n \to C$, where C is a (finite) set of class labels, models the human perception capability, then a neural network classifier is a function $\hat{f}(x)$ which approximates $f(x)$ from M training examples $\{(x^i, c^i)\}_{i=1,..,M}$. For example, a perception module of a self-driving car may input an image from a camera and must correctly classify the type of object in its view, irrespective of aspects such as the angle of its vision and image imperfections. Therefore, though they clearly include imperfections, all four pairs of images in Figure 1 should arguably be classified as automobiles, since they appear so to a human eye. Classifiers employed in vision tasks are typically multi-layer networks, which are high-dimensional, often with millions of dimensions, non-linear and potentially discontinuous: even a small network, such as that trained to classify hand-written images of digits 0-9, has over 60,000 real-valued

* This work is supported by the EPSRC Programme Grant on Mobile Autonomy. Part of this work was done while MK was visiting the Simons Institute for the Theory of Computing.

parameters and 21,632 neurons (dimensions) in its first layer. To increase the probability of correctly classifying a previously unseen image, regularisation techniques such as dropout are typically used, which improves the smoothness of the classifiers, in the sense that images that are close (within ϵ distance) of a training point are assigned the same class label.

Fig. 1. Automobile images (classified correctly) and their perturbed images (classified wrongly)

Unfortunately, it has been observed in [26] that deep neural networks, including highly trained and smooth networks optimised for vision tasks, are unstable with respect to so called *adversarial perturbations*. Such adversarial perturbations are (minimal) changes to the input image, often imperceptible to the human eye, that cause the network to misclassify the image. Examples include not only artificially generated random perturbations, but also (more worryingly) modifications of camera images [16,21] that correspond to resizing, cropping or change in lighting conditions. Figure 1 gives adversarial perturbations of automobile images that are misclassified as a bird, frog, airplane or horse by a highly trained state-of-the-art network. This obviously raises potential safety concerns for applications such as autonomous driving and calls for automated verification techniques that can verify the correctness of the classifiers.

The main difficulty with image classification tasks is that they do not have a formal specification in the usual sense: ideally, the performance of a classifier should match the perception ability and class labels assigned by a human. Traditionally, the correctness of a neural network classifier is expressed in terms of *risk* [27], defined as the probability of misclassification of a given image, weighted with respect to the input distribution μ of images. More recently, the notion of *robustness* of a network to adversarial perturbations was studied in [14]. They define robustness at a point x as the norm of a minimal perturbation that achieves a misclassification, from which they derive a measure of robustness for a classifier, independent from the data point, by taking the expectation over the input distribution μ. The tool DeepFool [19] implements a fast method to compute minimal adversarial examples, but the technique, although improving over [26], is *approximate* and does not provide convergence guarantees, nor a guarantee that the perturbed image will be misclassified. The computed adversarial examples can then be used to fine-tune the network, hence improving its robustness. Pointwise robustness is further studied in [10], where a more accurate approximate method to compute a robustness measure is introduced based on constraint solving.

Contributions. In this paper, we focus on safety of image classifiers and develop the first SMT-based automated verification framework for (deep) feed-forward multilayer neural networks. As in [14,10], our method assumes the existence of a (possibly infinite) region η around a data point x such that all points in the region have the same class. A key concept that the framework relies on is a so called *manipulation*, a map-

ping δ between points in the vector space that intuitively corresponds to a modification of an image, which is called minimal if it suffices to check for misclassification only at the end points, i.e. x and $\delta(x)$. Such manipulations can represent, for example, image deformations or camera imprecisions. We work layer by layer and define a network to be *safe* for input x and region η_k of layer k with respect to the set of manipulations Δ_k if applying the manipulations on x will not result in a class change for η_k. We employ discretisation to enable a *finite exhaustive* search of the high-dimensional region η_k for adversarial misclassifications. The discretisation approach is justified since the images are typically represented as vectors of discrete pixels (vectors of 8 bit RGB colours). In contrast to [19,26,10], our method can *guarantee* that adversarial examples are found for the given region η_k and Δ_k, assuming minimality of manipulations. If found, adversarial examples can be shown to human testers and/or used to fine-tune the network, and otherwise safety has been demonstrated for the chosen region and manipulations.

We implement the techniques using Z3 [7] and evaluate them on state-of-the-art networks, including regularised and deep learning networks. This includes image classification networks trained for classifying hand-written images of digits 0-9 (MNIST), 10 classes of small colour images (CIFAR10), and 1000 classes of colour images used for the well-known imageNet large-scale visual recognition challenge (ILSVRC) [3]. The perturbed images in Figure 1 are found automatically using our tool for the network trained on the CIFAR10 dataset.

2 Background on Neural Networks

We consider feed-forward multi-layer neural networks [11], henceforth abbreviated as neural networks. Perceptrons (neurons) in a neural network are arranged in disjoint layers, with each perceptron in one layer connected to the next layer, but no connection between perceptrons in the same layer. Each layer L_k of a network is associated with an n_k-dimensional vector space $D_{L_k} \subseteq \mathbb{R}^{n_k}$, in which each dimension corresponds to a perceptron. We write P_k for the set of perceptrons in layer L_k and $n_k = |P_k|$ is the number of perceptrons (dimensions) in layer L_k.

Formally, a *(feed-forward and deep) neural* network N is a tuple (L, T, Φ), where $L = \{L_k \mid k \in \{0, ..., n\}\}$ is a set of layers such that layer L_0 is the *input* layer and L_n is the *output* layer, $T \subseteq L \times L$ is a set of sequential connections between layers such that, except for the input and output layers, each layer has an incoming connection and an outgoing connection, and $\Phi = \{\phi_k \mid k \in \{1, ..., n\}\}$ is a set of *activation functions* $\phi_k : D_{L_{k-1}} \rightarrow D_{L_k}$, one for each non-input layer. Layers other than input and output layers are called the *hidden* layers.

The network is fed an input x (point in D_{L_0}) through its input layer, which is then propagated through the layers by successive application of the activation functions. An *activation* for point x in layer k is the value of the corresponding function, denoted $\alpha_{x,k} = \phi_k(\phi_{k-1}(...\phi_1(x))) \in D_{L_k}$, where $\alpha_{x,0} = x$. For perceptron $p \in P_k$ we write $\alpha_{x,k}(p)$ for the value of its activation on input x. For every activation $\alpha_{x,k}$ and layer $k' < k$, we define $Pre_{k'}(\alpha_{x,k}) = \{\alpha_{y,k'} \in D_{L_{k'}} \mid \alpha_{y,k} = \alpha_{x,k}\}$ to be the set of activations in layer k' whose corresponding activation in layer L_k is $\alpha_{x,k}$. The classification decision is made based on the activations in the output layer by, e.g., assigning to x the class

$\arg\max_{p \in P_n} \alpha_{x,n}(p)$. For simplicity, we use $\alpha_{x,n}$ to denote the class assigned to input x, and thus $\alpha_{x,n} = \alpha_{y,n}$ expresses that two inputs x and y have the same class.

The neural network classifier N represents a function $\hat{f}(x)$ which approximates $f(x) : D_{L_0} \to C$, a function that models the human perception capability in labelling images with labels from C, from M training examples $\{(x^i, c^i)\}_{i=1,...,M}$. Image classification networks, for example convolutional networks, may contain many layers, which can be non-linear, and work in high dimensions, which for the image classification problems can be of the order of millions. Digital images are represented as 3D tensors of pixels (width, height and depth, the latter to represent colour), where each pixel is a discrete value in the range 0..255. The training process determines real values for weights used as filters that are convolved with the activation functions. Since it is difficult to approximate f with few samples in the sparsely populated high-dimensional space, to increase the probability of classifying correctly a previously unseen image, various regularisation techniques such as dropout are employed. They improve the smoothness of the classifier, in the sense that points that are ϵ-close to a training point (potentially infinitely many of them) classify the same.

In this paper, we work with the code of the network and its trained weights.

3 The Verification Framework

In this section we define our notion of safety for a neural network, based on the concept of a manipulation of an image, and propose a verification framework. The main challenge is to ensure finite exhaustive coverage in presence of high-dimensionality and non-linearity, which we address by developing a layer-by-layer-analysis approach.

Safety and Robustness Our method assumes the existence of a (possibly infinite) region η around a data point x such that all points in the region have the same true class, as classified by a human. This is difficult, if not impossible, to define precisely, and instead approximate means to identify such a region and its diameter d are sought [14]. A network $\hat{f}$ approximating human capability f is said to be *not robust* at x [14] if there exists a point y in the region $\eta = \{z \in D_{L_0} \mid \|z - x\| \leq d\}$ of the input layer such that $\hat{f}(x) \neq \hat{f}(y)$. The point y, at a minimal distance from x, is known as an *adversarial example*. Our definition follows the same intuition, except that we work layer by layer, and therefore will identify such a region η_k, a subspace of D_{L_k}, at each layer L_k, for $k \in \{0, ..., n\}$, and successively refine the regions through the deeper layers. We justify this choice based on the observation [9,17,18] that deep neural networks are thought to compute progressively more powerful invariants as the depth increases. In other words, they gradually transform images into a representation in which the classes are separable by a linear classifier.

Assumption 1 *For each activation $\alpha_{x,k}$ of point x in layer L_k, the region $\eta_k(\alpha_{x,k})$ contains activations that the human observer believes to be so close to $\alpha_{x,k}$ that they should be classified the same as x.*

Based on this assumption, we have the following definition of safety. Intuitively, safety for network N at a point x means that the classification is robust at x against perturba-

tions within the region $\eta_k(\alpha_{x,k})$. Note that, while the perturbation is applied in layer L_k, the classification decision is based on the activation in the output layer L_n.

Definition 1. *[General Safety] Let $\eta_k(\alpha_{x,k})$ be a region in layer L_k of a neural network N such that $\alpha_{x,k} \in \eta_k(\alpha_{x,k})$. We say that N is* safe for input x and region $\eta_k(\alpha_{x,k})$*, written as $N, \eta_k \models x$, if for all activations $\alpha_{y,k}$ in $\eta_k(\alpha_{x,k})$ we have $\alpha_{y,n} = \alpha_{x,n}$.*

We remark that, unlike the notions of risk [27] and robustness [14,10], we work with safety for a specific point and do not account for the input distribution, but such expectation measures can be considered; see Section 6 for comparison.

Manipulations The main challenge for verification is the fact that the region η_k contains potentially an uncountable number of activations. Our approach relies on discretisation in order to enable a finite exploration of the region to discover and/or rule out adversarial perturbations. The key concept to achieve this is a *manipulation*, an operator $\delta_k : D_{L_k} \to D_{L_k}$ over the activations that intuitively models image perturbations, for example bad angles, scratches or weather conditions. More specifically, applying a manipulation $\delta_k(\alpha_{x,k})$ to an activation $\alpha_{x,k}$ will result in another activation such that the values of *some or all* dimensions are changed. We therefore represent a manipulation as a hyper-rectangle, defined for two activations $\alpha_{x,k}$ and $\alpha_{y,k}$ of layer L_k by $rec(\alpha_{x,k}, \alpha_{y,k}) = \times_{p \in P_k}[min(\alpha_{x,k}(p), \alpha_{y,k}(p)),\ max(\alpha_{x,k}(p), \alpha_{y,k}(p))]$.

For an activation $\alpha_{x,k}$ and a set Δ of manipulations, we denote by $rec(\Delta, \alpha_{x,k})$ the polyhedron which includes all hyper-rectangles that result from applying some manipulation in Δ on $\alpha_{x,k}$, i.e., $rec(\Delta, \alpha_{x,k}) = \bigcup_{\delta \in \Delta} rec(\alpha_{x,k}, \delta(\alpha_{x,k}))$. Let Δ_k be the set of all possible manipulations for layer L_k. To ensure region coverage, we define *valid* manipulation as follows.

Definition 2. *Given an activation $\alpha_{x,k}$, a set of manipulations $V(\alpha_{x,k}) \subseteq \Delta_k$ is* valid *if $\alpha_{x,k}$ is an interior point of $rec(V(\alpha_{x,k}), \alpha_{x,k})$, i.e., $\alpha_{x,k}$ is in $rec(V(\alpha_{x,k}), \alpha_{x,k})$ and does not belong to the boundary of $rec(V(\alpha_{x,k}), \alpha_{x,k})$.*

Figure 2 presents an example of valid manipulations in two-dimensional space: each arrow represents a manipulation, each dashed box represents a (hyper-)rectangle of the corresponding manipulation, and activation $\alpha_{x,k}$ is an interior point of the space from the dashed boxes.

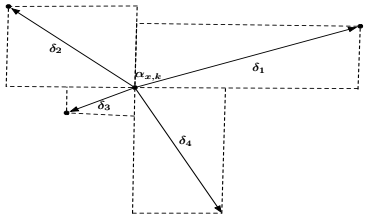


Fig. 2. Example of a set $\{\delta_1, \delta_2, \delta_3, \delta_4\}$ of valid manipulations in a 2-dimensional space

The choice of the type of manipulation is dependent on the application and user-defined, reflecting knowledge of the classification problem to model perturbations that

should or should not be allowed. Since we work with discretised spaces, which is a reasonable assumption for images, we introduce the notion of a *minimal* manipulation. If applying a minimal manipulation, it suffices to check for misclassification just at the end points, that is, $\alpha_{x,k}$ and $\delta_k(\alpha_{x,k})$. This allows an exhaustive, albeit impractical, exploration of the region in unit steps.

A manipulation $\delta_k^1(\alpha_{y,k})$ is finer than $\delta_k^2(\alpha_{x,k})$, written as $\delta_k^1(\alpha_{y,k}) \leq \delta_k^2(\alpha_{x,k})$, if any activation in the hyper-rectangle of the former is also in the hyper-rectangle of the latter. It is implied in this definition that $\alpha_{y,k}$ is an activation in the hyper-rectangle of $\delta_k^2(\alpha_{x,k})$. Moreover, we write $\delta_{k,k'}(\alpha_{x,k})$ for $\phi_{k'}(...\phi_{k+1}(\delta_k(\alpha_{x,k})))$, representing the corresponding activation in layer $k' \geq k$ after applying manipulation δ_k on the activation $\alpha_{x,k}$, where $\delta_{k,k}(\alpha_{x,k}) = \delta_k(\alpha_{x,k})$.

Definition 3. *A manipulation δ_k on an activation $\alpha_{x,k}$ is* minimal *if there does not exist manipulations δ_k^1 and δ_k^2 and an activation $\alpha_{y,k}$ such that $\delta_k^1(\alpha_{x,k}) \leq \delta_k(\alpha_{x,k})$, $\alpha_{y,k} = \delta_k^1(\alpha_{x,k})$, $\delta_k(\alpha_{x,k}) = \delta_k^2(\alpha_{y,k})$, and $\alpha_{y,n} \neq \alpha_{x,n}$ and $\alpha_{y,n} \neq \delta_{k,n}(\alpha_{x,k})$.*

Intuitively, a minimal manipulation does not have a finer manipulation that results in a different classification. However, it is possible to have different classifications before and after applying the minimal manipulation, i.e., it is possible that $\delta_{k,n}(\alpha_{x,k}) \neq \alpha_{x,n}$. It is not hard to see that the minimality of a manipulation implies that the class change in its associated hyper-rectangle can be detected by checking the class of the end points $\alpha_{x,k}$ and $\delta_k(\alpha_{x,k})$.

For simplicity, we work with the Euclidean norm to measure the distance between an image and its perturbation through δ_k, but the techniques generalise to other norms discussed in [14,15,10].

Bounded Variation Recall that we apply manipulations in layer L_k, but check the classification decisions in the output layer. To ensure *finite, exhaustive* coverage of the region, we introduce a continuity assumption on the mapping from space D_{L_k} to the output space D_{L_n}, adapted from the concept of bounded variation [8]. Given an activation $\alpha_{x,k}$ with its associated region $\eta_k(\alpha_{x,k})$, we define a "ladder" on $\eta_k(\alpha_{x,k})$ to be a set ld of activations containing $\alpha_{x,k}$ and finitely many, possibly zero, activations from $\eta_k(\alpha_{x,k})$. The activations in a ladder can be arranged into an increasing order $\alpha_{x,k} = \alpha_{x_0,k} < \alpha_{x_1,k} < ... < \alpha_{x_j,k}$ such that every activation $\alpha_{x_t,k} \in ld$ appears once and has a successor $\alpha_{x_{t+1},k}$ such that $\alpha_{x_{t+1},k} = \delta_k(\alpha_{x_t,k})$ for some manipulation $\delta_k \in V(\alpha_{x_t,k})$. For the greatest element $\alpha_{x_j,k}$, its successor should be outside the region $\eta_k(\alpha_{x,k})$, i.e., $\alpha_{x_{j+1},k} \notin \eta_k(\alpha_{x,k})$. Given a ladder ld, we write $ld(t)$ for its $t+1$-th activation, $ld[0..t]$ for the prefix of ld up to the $t+1$-th activation, and $last(ld)$ for the greatest element of ld. Figure 3 gives a diagrammatic explanation on the ladders.

Definition 4. *Let $\mathcal{L}(\eta_k(\alpha_{x,k}))$ be the set of ladders in $\eta_k(\alpha_{x,k})$. Then the* total variation *of the region $\eta_k(\alpha_{x,k})$ on the neural network with respect to $\mathcal{L}(\eta_k(\alpha_{x,k}))$ is*

$$V(N;\eta_k(\alpha_{x,k})) = \sup_{ld \in \mathcal{L}(\eta_k(\alpha_{x,k}))} \sum_{\alpha_{x_t,k} \in ld \setminus \{last(ld)\}} \mathrm{diff}_n(\alpha_{x_t,n}, \alpha_{x_{t+1},n})$$

where $\mathrm{diff}_n : D_{L_n} \times D_{L_n} \to \{0,1\}$ *is given by* $\mathrm{diff}_n(\alpha_{x,n}, \alpha_{y,n}) = 0$ *if* $\alpha_{x,n} = \alpha_{y,n}$ *and 1 otherwise. We say that the region $\eta_k(\alpha_{x,k})$ is a* bounded variation *if $V(N;\eta_k(\alpha_{x,k})) < \infty$,*

and are particularly interested in the case when $V(N; r_k(\alpha_{y,k})) = 0$, which is called a 0-variation.

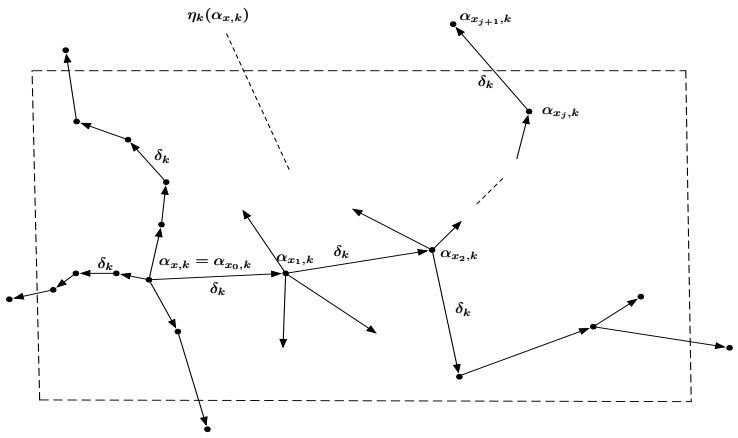


Fig. 3. Examples of ladders in region $\eta_k(\alpha_{x,k})$. Starting from $\alpha_{x,k} = \alpha_{x_0,k}$, the activations $\alpha_{x_1,k}...\alpha_{x_j,k}$ form a ladder such that each consecutive activation results from some valid manipulation δ_k applied to a previous activation, and the final activation $\alpha_{x_j,k}$ is outside the region $\eta_k(\alpha_{x,k})$.

The set $\mathcal{L}(\eta_k(\alpha_{x,k}))$ is *complete* if, for any ladder $ld \in \mathcal{L}(\eta_k(\alpha_{x,k}))$ of $j+1$ activations, any element $ld(t)$ for $0 \leq t \leq j$, and any manipulation $\delta_k \in V(ld(t))$, there exists a ladder $ld' \in \mathcal{L}(\eta_k(\alpha_{x,k}))$ such that $ld'[0..t] = ld[0..t]$ and $ld'(t+1) = \delta_k(ld(t))$. Intuitively, a complete ladder is a complete tree, on which each node represents an activation and each branch of a node corresponds to a valid manipulation. From the root $\alpha_{x,k}$, every path of the tree leading to a leaf is a ladder. Moreover, the set $\mathcal{L}(\eta_k(\alpha_{x,k}))$ is *covering* if the polyhedra of all activations in it cover the region $\eta_k(\alpha_{x,k})$, i.e.,

$$\eta_k(\alpha_{x,k}) \subseteq \bigcup_{ld \in \mathcal{L}(\eta_k(\alpha_{x,k}))} \bigcup_{\alpha_{x_t,k} \in ld \setminus \{last(ld)\}} rec(V(\alpha_{x_t,k}), \alpha_{x_t,k}). \tag{1}$$

Based on the above, we have the following definition of safety with respect to a set of manipulations. Intuitively, we *iteratively* and *nondeterministically* apply manipulations to explore the region $\eta_k(\alpha_{x,k})$, and safety means that no class change is observed by successive application of such manipulations.

Definition 5. *[Safety wrt Manipulations] Given a neural network N, an input x and a set Δ_k of manipulations, we say that N is* safe for input *x* with respect to the region η_k and manipulations Δ_k, *written as $N, \eta_k, \Delta_k \models x$, if the region $\eta_k(\alpha_{x,k})$ is a 0-variation for the set $\mathcal{L}(\eta_k(\alpha_{x,k}))$ of its ladders, which is complete and covering.*

It is straightforward to note that general safety in the sense of Definition 1 implies safety wrt manipulations, in the sense of Definition 5.

Theorem 1. *Given a neural network N, an input x, and a region η_k, we have that $N, \eta_k \models x$ implies $N, \eta_k, \Delta_k \models x$ for any set of manipulations Δ_k.*

In the opposite direction, we require the minimality assumption on manipulations.

Theorem 2. *Given a neural network N, an input x, a region $\eta_k(\alpha_{x,k})$ and a set Δ_k of manipulations, we have that $N, \eta_k, \Delta_k \models x$ implies $N, \eta_k \models x$ if the manipulations in Δ_k are minimal.*

Theorem 2 means that, under the minimality assumption over the manipulations, an *exhaustive* search through the complete and covering ladder tree from $\mathcal{L}(\eta_k(\alpha_{x,k}))$ can find adversarial examples, if any, and enable us to conclude that the network is safe at a given point if none are found. Though computing minimal manipulations is not practical, in discrete spaces by iterating over increasingly *refined* manipulations we are able to rule out the existence of adversarial examples in the region. This contrasts with *partial* exploration according to, e.g., [19,10]; for comparison see Section 6.

Layer by Layer Analysis Now we consider how to propagate the analysis layer by layer. To facilitate such analysis, in addition to the activation function $\phi_k : D_{L_{k-1}} \to D_{L_k}$ we also require a mapping $\psi_k : D_{L_k} \to D_{L_{k-1}}$ in the opposite direction, to represent how a manipulated activation of layer L_k affects the activations of layer L_{k-1}. We can simply take ψ_k as the inverse function of ϕ_k. In order to propagate safety of regions $\eta_k(\alpha_{x,k})$ at a point x into deeper layers, we assume the existence of functions η_k that map activations to regions, and impose the following restrictions on the functions ϕ_k and ψ_k, shown diagrammatically in Figure 4.

Definition 6. *The functions $\{\eta_0, \eta_1, ..., \eta_n\}$ and $\{\psi_1, ..., \psi_n\}$ mapping activations to regions are such that*

1. $\eta_k(\alpha_{x,k}) \subseteq D_{L_k}$, *for* $k = 0, ..., n$,
2. $\alpha_{x,k} \in \eta_k(\alpha_{x,k})$, *for* $k = 0, ..., n$, *and*
3. $\eta_{k-1}(\alpha_{i,k-1}) \subseteq \psi_k(\eta_k(\alpha_{x,k}))$ *for all* $k = 1, ..., n$.

Intuitively, the first two conditions state that each function η_k assigns a region around the activation $\alpha_{x,k}$, and the last condition that mapping the region η_k from layer L_k to L_{k-1} via ψ_k should cover the region η_{k-1}. The aim is to compute functions $\eta_{k+1}, ..., \eta_n$ based on η_k and the neural network.

The size and complexity of a deep neural network generally means that determining whether a given set Δ_k of manipulations is minimal is intractable. To partially counter this, we define a *refinement* relation between safety wrt manipulations for consecutive layers in the sense that $N, \eta_k, \Delta_k \models x$ is a refinement of $N, \eta_{k-1}, \Delta_{k-1} \models x$ if all manipulations δ_{k-1} in Δ_{k-1} are refined by a sequence of manipulations δ_k from the set Δ_k. Therefore, although we cannot theoretically confirm the minimality of Δ_k, they are refined layer by layer and, in discrete settings, this process can be bounded from below by the unit step. Moreover, we can work gradually from a specific layer inwards until an adversarial example is found, finishing processing when reaching the output layer, subject to the amount of computational resources available.

The refinement framework is given in Figure 5. The arrows represent the implication

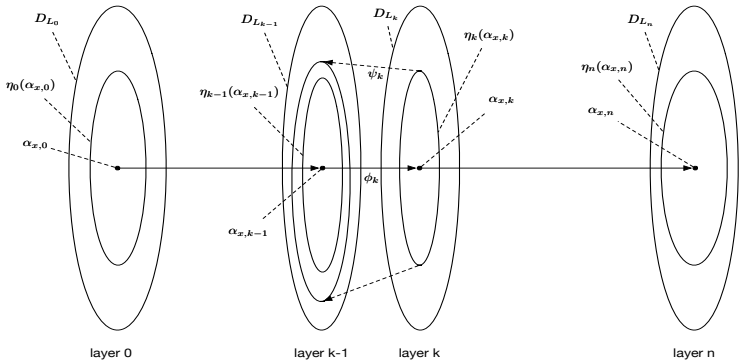


Fig. 4. Layer by layer analysis according to Definition 6

relations between the safety notions and are labelled with conditions if needed. The goal of the refinements is to find a chain of implications to justify $N, \eta_0 \models x$. The fact that $N, \eta_k \models x$ implies $N, \eta_{k-1} \models x$ is due to the constraints in Definition 6 when $\psi_k = \phi_k^{-1}$. The fact that $N, \eta_k \models x$ implies $N, \eta_k, \Delta_k \models x$ follows from Theorem 1. The implication from $N, \eta_k, \Delta_k \models x$ to $N, \eta_k \models x$ under the condition that Δ_k is minimal is due to Theorem 2.

We now define the notion of refinability of manipulations between layers. Intuitively, a manipulation in layer L_{k-1} is refinable in layer L_k if there exists a sequence of manipulations in layer L_k that implements the manipulation in layer L_{k-1}.

Definition 7. *A manipulation $\delta_{k-1}(\alpha_{y,k-1})$ is* refinable *in layer L_k if there exist activations $\alpha_{x_0,k}, ..., \alpha_{x_j,k} \in D_{L_k}$ and valid manipulations $\delta_k^1 \in V(\alpha_{x_0,k}), ..., \delta_k^j \in V(\alpha_{x_{j-1},k})$ such that $\alpha_{y,k} = \alpha_{x_0,k}$, $\delta_{k-1,k}(\alpha_{y,k-1}) = \alpha_{x_j,k}$, and $\alpha_{x_t,k} = \delta_k^t(\alpha_{x_{t-1},k})$ for $1 \leq t \leq j$. Given a neural network N and an input x, the manipulations Δ_k are a* refinement by layer *of η_{k-1}, Δ_{k-1} and η_k if, for all $\alpha_{y,k-1} \in \eta_{k-1}(\alpha_{z,k-1})$, all its valid manipulations $\delta_{k-1}(\alpha_{y,k-1})$ are refinable in layer L_k.*

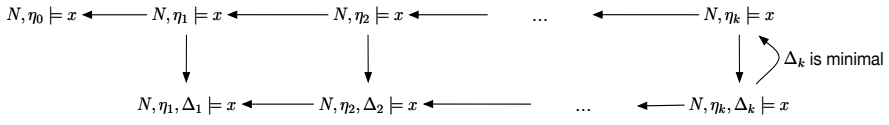


Fig. 5. Refinement framework

We have the following theorem stating that the refinement of safety notions is implied by the "refinement by layer" relation.

Theorem 3. *Assume a neural network N and an input x. For all layers $k \geq 1$, if manipulations Δ_k are refinement by layer of η_{k-1}, Δ_{k-1} and η_k, then we have that $N, \eta_k, \Delta_k \models x$ implies $N, \eta_{k-1}, \Delta_{k-1} \models x$.*

We note that any adversarial example of safety wrt manipulations $N, \eta_k, \Delta_k \models x$ is also an adversarial example for general safety $N, \eta_k \models x$. However, an adversarial example $\alpha_{x,k}$ for $N, \eta_k \models x$ at layer k needs to be checked to see if it is an adversarial example of $N, \eta_0 \models x$, i.e. for the input layer. Recall that $Pre_{k'}(\alpha_{x,k})$ is not necessarily unique. This is equivalent to checking the emptiness of $Pre_0(\alpha_{x,k}) \cap \eta_0(\alpha_{x,0})$. If we start the analysis with a hidden layer $k > 0$ and there is no specification for η_0, we can instead consider checking the emptiness of $\{\alpha_{y,0} \in Pre_0(\alpha_{x,k}) \mid \alpha_{y,n} \neq \alpha_{x,n}\}$.

While Theorem 1 and 2 provide a *finite* way to verify safety of a deep neural network against adversarial manipulations, the high-dimensionality of the region $\eta_k(\alpha_{x,k})$ can make any computational approach impractical. In our implementation, we use the concept of a *feature* to divide the region $\eta_k(\alpha_{x,k})$ into a set of features, and exploit their independence and low-dimensionality, see Appendix C. It has been argued, in e.g. [13] for natural images, that natural data, for example natural images and sound, forms a high-dimensional manifold, which embeds tangled manifolds to represent their features. Feature manifolds usually have lower dimension than the data manifold, and a classification algorithm is to separate a set of tangled manifolds. By assuming that the appearance of features is independent, we can manipulate them one by one regardless of the manipulation order, and thus reduce the problem of size $O(2^{d_1+\ldots+d_m})$ into a set of smaller problems of size $O(2^{d_1}), ..., O(2^{d_m})$.

4 Implementation

The theory developed in the previous section can be summarised as a verification procedure given below.

Algorithm 1 *Given a neural network N and an input x, recursively perform the following steps, starting from some layer $l \geq 0$. Let $k \geq l$ be the current layer under consideration.*

1. *find a region η_k such that if $k > l$ then η_k and η_{k-1} satisfy Definition 6;*
2. *find a manipulation set Δ_k such that if $k > l$ then Δ_k is a refinement by layer of η_{k-1}, Δ_{k-1} and η_k according to Definition 7;*
3. *verify whether $N, \eta_k, \Delta_k \models x$,*
   (a) *if $N, \eta_k, \Delta_k \models x$ then*
      i. *report that N is safe at x with respect to $\eta_k(\alpha_{x,k})$ and Δ_k, and*
      ii. *continue to layer $k + 1$;*
   (b) *if $N, \eta_k, \Delta_k \not\models x$, then report an adversarial example.*

The procedure terminates when reaching a specific layer or a timeout bound.

The main challenge in the above is identifying the region η_k and sets of manipulations Δ_k, which intuitively correspond to perturbations of the image such as scratches and imperfections. In our implementation, these are determined automatically, see Appendix B, but they can be specified by the user for the classification problem at hand.

Similarly to [26,14,19], we consider the Euclidean distance between the image x and its perturbation, except that we compute the distance between activations in the *hidden* layers, rather than images in the input layer, and can deal with more refined

measurements. The methods can be extended to other norms used in [15,10]. For the choice of manipulations, since we work with features (see Appendix C), we select a subset of dimensions (perceptrons) according to a simple heuristic, namely, we select those dimensions that are far away from the average activation value of the layer. This is critical to obtain reasonable performance on state-of-the art networks, and we find that for networks with thousands of dimensions an adversarial example can often be found quickly by considering only a dozen of dimensions in the hidden layer. The justification for our choice of heuristics is that the knowledge represented by activations in deeper layers is more explicit, as the networks are thought to increase in regularity as the depth increases.

We implement Algorithm 1 by utilising satisfiability modulo theory (SMT) solvers. The SMT problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. For checking refinement by layer, we use the theory of linear real arithmetic with existential and universal quantifiers, and for verification within a layer (0-variation) we use the same theory but without universal quantification. The details of the encoding and the approach taken to compute the regions and manipulations are described in Appendix B. We note that we work with a single point rather than activation functions. In contrast, [22] approximate the sigmoid functions using constraints, which is not scalable, and [10] work instead with the ReLU functions (i.e., thresholding to zero) and provide an approximate solution based on linear programming.

5 Experimental Results

The proposed framework has been implemented as a software tool called DLV[1] (abbrv. Deep Learning Verification) written in Python. The SMT solver we use is Z3 [7], which has Python APIs. The neural networks are built from a widely-used neural networks library Keras [2] with a deep learning package Theano [5] as its backend.

We validate DLV on a set of experiments performed for neural networks trained for classification based on a predefined multi-dimensional surface (small size networks), as well as image classification (medium size networks). These networks respectively use two representative types of layers: fully connected layers and convolutional layers. They may also use other types of layers, e.g., the ReLU layer, the pooling layer, the zero-padding layer, and the dropout layer. The experiments are conducted on a MacBook Pro laptop, with 2.7 GHz Intel Core i5 CPU and 8 GB memory.

Two-Dimensional Point Classification Network To demonstrate the working of our framework, we consider a neural network trained for classifying points above and below a two-dimensional curve shown in red in Figure 6 and Figure 7. The network has three fully-connected hidden layers with the ReLU activation function. The input layer has two perceptrons, every hidden layer has 20 perceptrons, and the output layer has two perceptrons. The network is trained with 5,000 points sampled from the provided two-dimensional space, and has an accuracy of more than 99%.

[1] See `https://github.com/xiaoweih/DLV` and the Appendix for the parameter setting

For a given input $x = (3.59, 1.11)$, we start from the input layer and define a region around this point by taking unit steps in both directions

$$\eta_0(\alpha_{x,0}) = [3.59 - 1.0, 3.59 + 1.0] \times [1.11 - 1.0, 1.11 + 1.0] = [2.59, 4.59] \times [0.11, 2.11]$$

The manipulation set Δ_0 is shown in Figure 6: there are 9 points, of which the point in the middle represents the activation $\alpha_{x,0}$ and the other 8 points represent the activations resulting from applying one of the manipulations in Δ_0 on $\alpha_{x,0}$. Note that, although there are class changes in the region $\eta_0(\alpha_{x,0})$, the manipulation set Δ_0 is not able to detect such changes. Therefore, we have that $N, \eta_0, \Delta_0 \models x$.

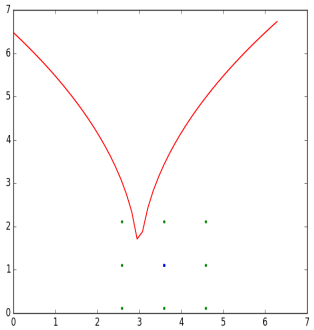


Fig. 6. Input layer

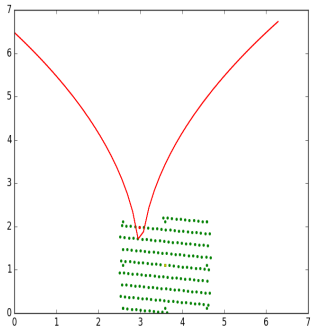


Fig. 7. First hidden layer

Now consider layer $k = 1$. To obtain the region $\eta_1(\alpha_{x,1})$, the tool selects two dimensions $p_{1,17}, p_{1,19} \in P_1$ in layer L_1 with indices 17 and 19 and computes

$$\eta_1(\alpha_{x,1}) = [\alpha_{x,1}(p_{1,17}) - 3.6, \alpha_{x,1}(p_{1,17}) + 3.6] \times [\alpha_{x,1}(p_{1,19}) - 3.52, \alpha_{x,1}(p_{1,19}) + 3.52]$$

The manipulation set Δ_1, after mapping back to the input layer with function ψ_1, is given as Figure 7. Note that η_1 and η_0 satisfy Definition 6, and Δ_1 is a refinement by layer of η_0, Δ_0 and η_1. We can see that a class change can be detected (represented as the red coloured point). Therefore, we have that $N, \eta_1, \Delta_1 \not\models x$.

Image Classification Network for the MNIST Handwritten Image Dataset The well-known MNIST image dataset contains images of size 28×28 and one channel and the network is trained with the source code given in [4]. The trained network is of medium size with 600,810 parameters, has an accuracy of more than 99%, and is state-of-the-art. It has 12 layers, within which there are 2 convolutional layers, as well as layers such as ReLU, dropout, fully-connected layers and a softmax layer. The images are preprocessed to make the value of each pixel within the bound $[0, 1]$.

Given an image x, we start with layer $k = 1$ and the parameter set to at most 150 dimensions (there are 21632 dimensions in layer L_1). All η_k, Δ_k for $k \geq 2$ are computed according to the simple heuristic mentioned in Section 4 and satisfy Definition 6 and Definition 7. For the region $\eta_1(\alpha_{x,1})$, we allow changes to the activation value of each

selected dimension that are within [-1,1]. The set Δ_1 includes manipulations that can change the activation value for a subset of the 150 dimensions, by incrementing or decrementing the value for each dimension by 1. The experimental results show that for most of the examples we can find a class change within 100 dimensional changes in layer L_1, by comparing the number of pixels that have changed, and some of them can have less than 30 dimensional changes. Figure 8 presents examples of such class changes for layer L_1. We also experiment on images with up to 40 dimensional changes in layer L_1; the tool is able to check the entire network, reaching the output layer and claiming that $N, \eta_k, \Delta_k \models x$ for all $k \geq 1$. While training of the network takes half an hour, finding an adversarial example takes up to several minutes.

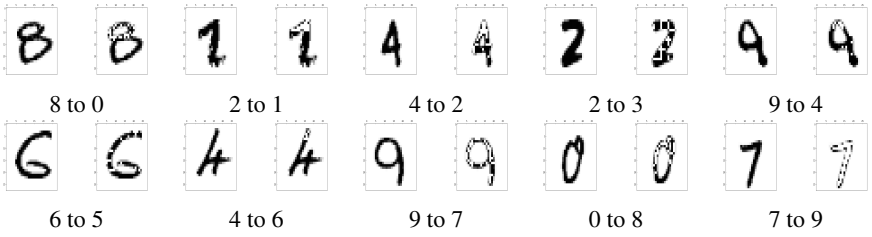


Fig. 8. Adversarial Examples for a Neural Network Trained on MNIST

Image Classification Network for the CIFAR-10 Small Image Dataset We work with a medium size neural network, trained with the source code from [1] for more than 12 hours on the well-known CIFAR10 dataset. The inputs to the network are images of size 32×32 with three channels. The trained network has 1,250,858 real-valued parameters and includes convolutional layers, ReLU layers, max-pooling layers, dropout layers, fully-connected layers, and a softmax layer.

As an illustration of the type of perturbations that we are investigating, consider the images in Figure 9, which correspond to the parameter setting of up to 25, 45, 65, 85, 105, 125, 145 dimensions, respectively, for layer $k = 1$. The manipulations change the activation values of these dimensions. Each image is obtained by mapping back from the first hidden layer and represents a point close to the boundary of the corresponding region. The relation $N, \eta_1, \Delta_1 \models x$ holds for the first 7 images, but fails for the last one and the image is classified as a truck. Intuitively, our choice of the region $\eta_1(\alpha_{x,1})$ identifies the subset of dimensions with most extreme activations, taking advantage of the analytical capability of the first hidden layer. A higher number of selected dimensions implies a larger region in terms of the Euclidean norm in which we apply manipulations, and, more importantly, suggests a more dramatic change to the knowledge represented by the activations when moving to the boundary of the region.

We also work with 500 dimensions and otherwise the same experimental parameters as for MNIST. Figure 11 in Appendix A gives 16 pairs of original images (classified correctly) and perturbed images (classified wrongly). We found that, while the manipulations lead to human-recognisable modifications to the images, the perturbed images can be classified wrongly by the network. For each image, finding an adversarial example ranges from seconds to 20 minutes.

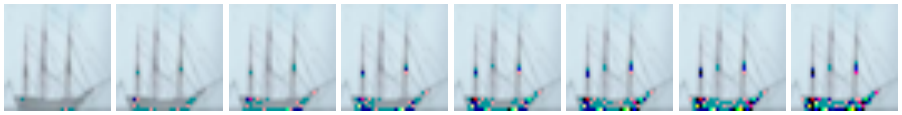


Fig. 9. An Illustrative Example from the Cifar-10 Dataset: the last image classifies as a truck.

Image Classification Network for the ImageNet Dataset We also conduct experiments on a large image classification network trained on the popular ImageNet dataset. The images are of size 224×224 and have three channels. The network is the model of the 16-layer network [25], called VGG16, used by the VGG team in the ILSVRC-2014 competition, downloaded from [6]. The trained network has 138,357,544 real-valued parameters and includes convolutional layers, ReLU layers, zero-padding layers, dropout layers, max-pooling layers, fully-connected layers, and a softmax layer. The experimental parameters are the same as for the previous two experiments, except that we work with 20,000 dimensions.

We again include several pairs of original and perturbed images in Figure 12 in Appendix A. In Figure 10 we also give two examples of street sign images. The image on the left is reported unsafe for the second layer with 6346 dimensional changes (0.2% of the 3,211,264 dimensions of layer L_2). The one on the right is reported safe for 20,000 dimensional changes of layer L_2. It appears that more complex manipulations, involving more dimensions (perceptrons), are needed in this case to cause a class change.

Fig. 10. Street Sign Images. Found an adversarial example for the left image (class changed into bird house), but cannot find an adversarial example for the right image for 20,000 dimensions.

6 Related Work

The need for verified AI is argued in, e.g., [24]. A verification approach for neural networks was proposed in [22], where, using the notation of this paper, safety is defined as the existence, for all inputs in a region $\eta_0 \in D_{L_0}$, of a corresponding output in another region $\eta_n \subseteq D_{L_n}$. They encode the entire network as a set of constraints, which can then be solved by a SAT solver, but their approach only works with 6 neurons (3 hidden neurons). A similar idea is presented in [23]. In contrast, we work layer by layer and obtain much greater scalability.

Concerns about the instability of neural networks to adversarial examples were first raised in [26], where optimisation is used to identify misclassifications. In our notation, they are using a *deterministic* manipulation $\delta(x) = x + \epsilon sign(\nabla_x J(x, \alpha_{x,n}))$, where x is an image in matrix representation, ϵ is a hyper-parameter that can be tuned to get different manipulated images, and $J(x, \alpha_{x,n})$ is the cross-entropy cost function of the neural network on input x and class $\alpha_{x,n}$. Therefore, their approach will test a set of discrete points in the region $\eta_0(\alpha_{x,0})$ of the input layer. A method for computing the perturbations is also proposed, which is based on box-constrained optimisation and is approximate in view of non-convexity of the search space. This work is followed by [16], which considers more general manipulations. Unlike [26], these manipulations allow *iterative* (and deterministic) manipulations, and therefore will test a lasso-type ladder tree (i.e., a ladder tree without branches) $\mathcal{L}(\eta_k(\alpha_{x,k}))$, which does not satisfy the covering property. In [20], instead of working with a single image, an evolutionary algorithm is employed for a population of images. For each individual image in the current population, the manipulation is the mutation and/or crossover. While mutations can be *nondeterministic*, the manipulations of an individual image are also following a lasso-type ladder tree which is not covering. We also mention that [28] uses several distortions such as JPEG compression, thumbnail resizing, random cropping, etc, to test the robustness of the trained network. These distortions can be understood as manipulations. All these methods are *black box* and do *not* guarantee that manipulated images will be misclassified [16], in contrast to this paper.

The notion of robustness studied in [14] has some similarities to our definition of safety, except that the authors work with values *averaged* over the input distribution μ, which is difficult to estimate accurately in high dimensions. As in [26,16], they use optimisation without convergence guarantees, as a result computing only an approximation to the minimal perturbation. In [10] pointwise robustness is adopted, which corresponds to our general safety; they also use a constraint solver but represent the full constraint system and propose an approximate LP method to achieve tractability. In contrast, we work directly with activations rather than an encoding of activation functions, and our method *exhaustively* searches through the complete ladder tree for an adversarial example by iterative and nondeterministic application of manipulations. Further, our definition of a manipulation is more flexible, since it allows us to select a *subset* of dimensions, and each such subset can have a different region diameter computed with respect to a different norm.

7 Conclusions

This paper presents an automated verification framework for checking safety of deep neural networks that is based on a systematic exploration of a region around a data point to search for adversarial manipulations of a given type, and propagating the analysis into deeper layers. Though we focus on the classification task, the approach also generalises to other types of networks. We have implemented the approach using SMT and validated it on several state-of-the-art neural network classifiers for realistic images. The results are encouraging, with adversarial examples found in some cases in a matter of seconds when working with few dimensions, but the verification process itself is expo-

nential in the number of features and has prohibitive complexity for larger images. The performance and scalability of our method can be significantly improved through parallelisation. It would be interesting to see if the notions of regularity suggested in [18] permit a symbolic approach, and whether an abstraction refinement framework can be formulated to improve the scalability and computational performance.

References

1. Cifar10 model for keras. https://github.com/fchollet/keras/blob/master/examples/cifar10_cnn.py.
2. Keras. https://keras.io.
3. Large scale visual recognition challenge. http://www.image-net.org/challenges/LSVRC/.
4. mnist cnn network. https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py.
5. Theano. http://deeplearning.net/software/theano/.
6. Vgg16 model for keras. https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3.
7. z3. http://rise4fun.com/z3.
8. Luigi Ambrosio, Nicola Fusco, and Diego Pallara. *Functions of bounded variation and free discontinuity problems*. Oxford Mathematical Monographs. Oxford University Press, 2000.
9. Fabio Anselmi, Joel Z. Leibo, Lorenzo Rosasco, Jim Mutch, Andrea Tacchetti, and Tomaso Poggio. Unsupervised learning of invariant representations. *Theoretical Computer Science*, 633:112–121, 2016.
10. Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. *CoRR*, abs/1605.07262, 2016. To appear in NIPS.
11. Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.
12. Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv:1604.07316*, 2016.
13. Gunnar E. Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *International Journal of Computer Vision*, 76(1), 2008.
14. Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. *CoRR*, abs/1502.02590, 2015.
15. Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.
16. Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv:1607.02533*, 2016.
17. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.
18. Stéphane Mallat. Understanding deep convolutional networks. *Philosohical Transactions of the Royal Society A*, 2016.
19. Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015.
20. Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Computer Vision and Pattern Recognition (CVPR '15)*, 2015.
21. Nicolas Papernot, Patrick Drew McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *CoRR*, abs/1602.02697, 2016.

22. Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *CAV 2010*, pages 243–257, 2010.
23. Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker. Towards verification of artificial neural networks. In *18th Workshop on Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen" (MBMV)*, pages 30–40, 2015.
24. Sanjit A. Seshia and Dorsa Sadigh. Towards verified artificial intelligence. *CoRR*, abs/1606.08514, 2016.
25. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.
26. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR-2014)*, 2014.
27. Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 831–838, 1991.
28. Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In *CVPR 2016*, 2016.

A Additional Adversarial Examples Found for Neural Nets

Fig. 11. Adversarial Examples for a Neural Network Trained on the CIFAR-10 Dataset

B Selection of Regions and Manipulations

The procedure in Algorithm 1 can start from any layer L_l in the network, which gives the user the flexibility to select the regions η_k and manipulations Δ_k for $k \geq l$. The tool implements an automated procedure to select these according to parameters $\{dims_l\} \cup V_l$, which are described below.

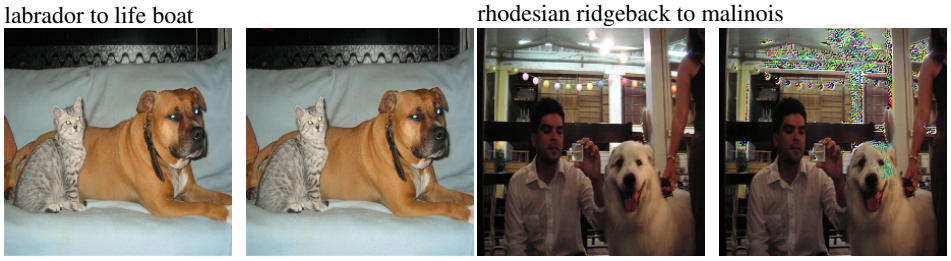
For the first layer to be considered, i.e., $k = l$, the region $\eta_k(\alpha_{x,k})$ is defined by first selecting the subset of $dims_k$ dimensions from P_k whose activation values are furthest away from the average activation value of the layer[2]. Intuitively, the knowledge represented by these activations is more explicit than the knowledge represented by the other dimensions, and manipulations over more explicit knowledge are more likely to result in a class change. Let $avg_k = (\sum_{p \in P_k} \alpha_{x,k}(p))/n_k$ be the average activation value of layer L_k. We let $dims_k(\eta_k(\alpha_{x,k}))$ be the first $dims_k$ dimensions $p \in P_k$ with the greatest values $|\alpha_{x,k}(p) - avg|$ among all dimensions, and then define

$$\eta_k(\alpha_{x,k}) = \times_{p \in dims_k(\eta_k(\alpha_{x,k}))} [\alpha_{x,k}(p) - s_p * m_p, \alpha_{x,k}(p) + s_p * m_p] \tag{2}$$

[2] We also considered other approaches, including computing derivatives up to several layers, but for the experiments we conduct they are less effective.


labrador to life boat

rhodesian ridgeback to malinois


boxer to rhodesian ridgeback

great pyrenees to kuvasz

Fig. 12. Adversarial Examples for the VGG16 Network Trained on the imageNet Dataset

i.e., a $dims_k$-polytope containing the activation $\alpha_{x,k}$, where s_p represents a small span and m_p represents the number of such spans. Let $V_k = \{s_p, m_p \mid p \in dims_k(\eta_k(\alpha_{x,k}))\}$ be a set of variables.

Let d be a function mapping from $dims_k(\eta_k(\alpha_{x,k}))$ to $\{-1, 0, +1\}$ such that $\{d(p) \neq 0 \mid p \in dims_k(\eta_k(\alpha_{x,k}))\} \neq \emptyset$. Let a manipulation δ_k^d be

$$\delta_k^d(\alpha_{y,k})(p) = \begin{cases} \alpha_{y,k}(p) - s_p & \text{if } d(p) = -1 \\ \alpha_{y,k}(p) & \text{if } d(p) = 0 \\ \alpha_{y,k}(p) + s_p & \text{if } d(p) = +1 \end{cases} \tag{3}$$

for activation $\alpha_{y,k} \in \eta_k(\alpha_{x,k})$. That is, each manipulation changes a subset of the dimensions by the span s_p, according to the directions given in d. The set Δ_k is defined by collecting the set of all such manipulations. Based on this, we can define a set $\mathcal{L}(\eta_k(\alpha_{x,k}))$ of ladders, which is complete and covering.

Determining the region η_k according to η_{k-1} Given $\eta_{k-1}(\alpha_{x,k-1})$ and the functions ϕ_k and ψ_k, the tool automatically finds a region $\eta_k(\alpha_{x,k})$ satisfying Definition 6 using the following approach. According to the function ϕ_k, the activation value $\alpha_{x,k}(p)$ of perceptron $p \in P_k$ is computed from activation values of a subset of perceptrons in P_{k-1}. We let $Vars(p) \subseteq P_{k-1}$ be such a set of perceptrons. The selection of dimensions in $dims_k(\eta_k(\alpha_{x,k}))$ depends on $dims_{k-1}(\eta_{k-1}(\alpha_{x,k-1}))$ and ϕ_k, by requiring that, for every $p' \in dims_{k-1}(\eta_{k-1}(\alpha_{x,k-1}))$, there is at least one dimension $p \in dims_k(\eta_k(\alpha_{x,k}))$ such that $p' \in Vars(p)$. In the tool, we let

$$dims_k(\eta_k(\alpha_{x,k})) = \{\arg\max_{p \in P_k}\{|\alpha_{x,k}(p) - avg_k| \mid p' \in Vars(p)\} \mid p' \in dims_{k-1}(\eta_{k-1}(\alpha_{x,k-1}))\} \tag{4}$$

Therefore, the restriction of Definition 6 can be expressed with the following formula:

$$\forall \alpha_{y,k-1} \in \eta_k(\alpha_{x,k-1}) : \alpha_{y,k-1} \in \psi_k(\eta_k(\alpha_{x,k})) \tag{5}$$

We omit the details of rewriting $\alpha_{y,k-1} \in \eta_k(\alpha_{x,k-1})$ and $\alpha_{y,k-1} \in \psi_k(\eta_k(\alpha_{x,k}))$ into boolean expressions, which follow from standard techniques. Note that this expression includes variables in V_k, V_{k-1} and $\alpha_{y,k-1}$. The variables in V_{k-1} are fixed for a given $\eta_{k-1}(\alpha_{x,k-1})$. Because such a region $\eta_k(\alpha_{x,k})$ always exists, a simple iterative procedure can be invoked to gradually increase the size of the region represented with variables in V_k to eventually satisfy the expression.

Determining the manipulation set Δ_k according to $\eta_k(\alpha_{x,k})$, $\eta_{k-1}(\alpha_{x,k-1})$, and Δ_{k-1}
The values of the variables V_k obtained from the satisfiability of Expression (5) yield a definition of manipulations using Expression (3). However, the obtained values for span variables s_p do not necessarily satisfy the "refinement by layer" relation as defined in Definition 7. Therefore, we need to adapt the values for the variables V_k while, at the same time, retaining the region $\eta_k(\alpha_{x,k})$. To do so, we could rewrite the constraint in Definition 7 into a formula, which can then be solved by an SMT solver. But, in practice, we notice that such *precise* computations easily lead to overly small spans s_p, which in turn result in an unacceptable amount of computation needed to verify the relation $N, \eta_k, \Delta_k \models x$.

To reduce computational cost, we work with a weaker "refinable in layer L_k" notion, parameterised with respect to precision. Given two activations $\alpha_{y,k}$ and $\alpha_{m,k}$, we use $dist(\alpha_{y,k}, \alpha_{m,k})$ to represent their Euclidean distance.

Definition 8. *A manipulation $\delta_{k-1}(\alpha_{y,k-1})$ is* refinable *in layer L_k with precision $\varepsilon > 0$ if there exists a sequence of activations $\alpha_{x_0,k}, ..., \alpha_{x_j,k} \in D_{L_k}$ and valid manipulations $\delta_k^1 \in V(\alpha_{x_0,k}), ..., \delta_k^d \in V(\alpha_{x_{j-1},k})$ such that $\alpha_{y,k} = \alpha_{x_0,k}$, $\delta_{k-1,k}(\alpha_{y,k-1}) \in rec(\alpha_{x_{j-1},k}, \alpha_{x_j,k})$, $dist(\alpha_{x_{j-1},k}, \alpha_{x_j,k}) \leq \epsilon$, and $\alpha_{x_t,k} = \delta_k^t(\alpha_{x_{t-1},k})$ for $1 \leq t \leq j$. Given a neural network N and an input x, the manipulations Δ_k are a* refinement by layer *of $\eta_k, \eta_{k-1}, \Delta_{k-1}$ with precision ε if, for all $\alpha_{y,k-1} \in \eta_{k-1}(\alpha_{x,k-1})$, all its legal manipulations $\delta_{k-1}(\alpha_{y,k-1})$ are refinable in layer L_k with precision ε.*

Comparing with Definition 7, the above definition replaces $\delta_{k-1,k}(\alpha_{y,k-1}) = \alpha_{x_j,k}$ with $\delta_{k-1,k}(\alpha_{y,k-1}) \in rec(\alpha_{x_{j-1},k}, \alpha_{x_j,k})$ and $dist(\alpha_{x_{j-1},k}, \alpha_{x_j,k}) \leq \varepsilon$. Intuitively, instead of requiring a manipulation to reach the activation $\delta_{k-1,k}(\alpha_{y,k-1})$ precisely, this definition allows for each $\delta_{k-1,k}(\alpha_{y,k-1})$ to be within the hyper-rectangle $rec(\alpha_{x_{j-1},k}, \alpha_{x_j,k})$. To find suitable values for V_k according to the approximate "refinement-by-layer" relation with precision, we use a variable h to represent the maximal number of manipulations of layer L_k used to express a manipulation in layer $k-1$. The tool automatically adapts the value of h (and variables s_p and n_p in V_k) to ensure the satisfiability of the following formula, which expresses the constraints of Definition 8.

$$\begin{aligned}&\forall \alpha_{y,k-1} \in \eta_k(\alpha_{x,k-1}) \forall d \forall \delta_{k-1}^d \in V_{k-1}(\alpha_{y,k-1}) \exists \alpha_{y_0,k}, ..., \alpha_{y_h,k} \in \eta_k(\alpha_{x,k}) : \alpha_{y_0,k} = \alpha_{y,k} \wedge \\ &\bigwedge_{t=0}^{h-1} \alpha_{y_{t+1},k} = \delta_k^d(\alpha_{y_t,k}) \wedge \bigvee_{t=0}^{h-1} (\delta_{k-1,k}^d(\alpha_{y,k}) \in rec(\alpha_{y_t,k}, \alpha_{y_{t+1},k}) \wedge dist(\alpha_{y_t,k}, \alpha_{y_{t+1},k}) \leq \varepsilon\end{aligned} \tag{6}$$

It is noted that s_p and m_p for $p \in dims_k(\eta_k(\alpha_{x,k}))$ are employed when expressing δ_k^d. The manipulation δ_k^d is obtained from δ_{k-1}^d by considering the corresponding relation between dimensions in $dims_k(\eta_k(\alpha_{x,k}))$ and $dims_{k-1}(\eta_{k-1}(\alpha_{x,k-1}))$. The precision ε is an input parameter for the tool.

C Features

When computing the relation $N, \eta_k, \Lambda_k \models x$, the region $\eta_k(\alpha_{x,k})$ can be high-dimensional. For example, in our experiments on the network for the CIFAR-10 dataset, we let $dims_k = 500$, and in the experiments on the network for ImageNet we let $dims_k = 20,000$. Such high dimensions can easily make any exhaustive search intractable if done in a naive way. In the following, we use the concept of a *feature* to partition the region $\eta_k(\alpha_{x,k})$ into a set of sub-regions called features. The features can be independent and of lower dimension, which leads to a practical algorithm for safety verification.

For each layer L_k, a feature function $f_k : D_{L_k} \to \mathcal{P}(D_{L_k})$ assigns a small region for each activation $\alpha_{x,k}$ in the space D_{L_k}, where $\mathcal{P}(D_{L_k})$ is the set of subspaces of D_{L_k}. The region $f_k(\alpha_{x,k})$ may have lower dimension than that of D_k. Intuitively, it defines for each point in the high-dimensional space D_{L_k} the most explicit feature it has. Such features may represent, e.g., the red-coloured frame of a street sign in Figure 10. It has been argued in, e.g., [13] for natural images that natural data, e.g., natural images and sound, etc, forms a high-dimensional manifold, which embeds tangled manifolds to represent their features. Feature manifolds usually have lower dimension than the data manifold, and the task of a neural network classification algorithm is to separate a set of tangled manifolds.

We remark that it is reasonable to assume that, in the high-dimensional spaces of an image classification network, *the features are independent*. Intuitively, for the input layer, an image usually has several main features, e.g., a human or a dog, and, for a dog, it has features such as the shape of eyes and ears. These features can be regarded as independent. For the hidden layers, due to the increase of linearity, features become more explicit and therefore independent. Moreover, defining the feature f_k on each activation as a single region corresponding to a specific feature is without loss of generality: although an activation such as an image may include multiple features, the independence relation between features suggests the existence of a total relation between these features. The function f_k essentially defines for each activation one particular feature, subject to certain criteria such as explicit knowledge.

The analysis of activations in hidden layers, as performed by our method, provides an opportunity to *discover the features automatically*. In the tool, the automatic discovery of features is achieved by the same heuristic that has been used to find the region $\eta_k(\alpha_{x,k})$, i.e., identifying a set of dimensions whose activation values are far away from the average value of the layer. After identifying the features, we can manipulate them one by one regardless of the manipulation order. By assuming that features have lower dimensions and different features include disjoint sets of dimensions, we can break down a problem of size $O(2^{d_1+...+d_m})$ into a sequential set of smaller problems of size $O(2^{d_1}), ..., O(2^{d_m})$, respectively.

More specifically, similarly to the region $\eta_k(\alpha_{x,k})$, every feature $f_k(\alpha_{y,k})$ is defined according to a pre-specified number $dims_{k,f}$ of dimensions, which is taken as an input to the tool. Let $dims_k(f_k(\alpha_{y,k}))$ be the set of dimensions selected according to the heuristic. Then we have that

$$f_k(\alpha_{y,k})(p) = \begin{cases} \eta_k(\alpha_{x,k})(p), & \text{if } p \in dims_k(f_k(\alpha_{y,k})) \\ [\alpha_{y,k}(p), \alpha_{y,k}(p)] & \text{otherwise.} \end{cases} \tag{7}$$

Moreover, we need a set of features to partition the region $\eta_k(\alpha_{x,k})$ as follows.

Definition 9. *A set $\{f_1, ..., f_m\}$ of regions is a partition of $\eta_k(\alpha_{x,k})$, written as $\pi(\eta_k(\alpha_{x,k}))$, if $dims_{k,f}(f_i) \cap dims_{k,f}(f_j) = \emptyset$ for $i, j \in \{1, ..., m\}$ and $\eta_k(\alpha_{x,k}) = \times_{i=1}^{m} f_i$.*

Given such a partition $\pi(\eta_k(\alpha_{x,k}))$, we define a function $acts(x, k)$ by

$$acts(x, k) = \{\alpha_{y,k} \in x \mid x \in \pi(\eta_k(\alpha_{x,k}))\} \tag{8}$$

which contains one point for each feature. Then, we reduce the checking of 0-variation of a region $\eta_k(\alpha_{x,k})$ to the following problems:

- checking whether the points in $acts(x, k)$ have the same class as $\alpha_{x,k}$, and
- checking the 0-variation of all features in $\pi(\eta_k(\alpha_{x,k}))$.

Computation of $Pre_0(\alpha_{y,k})$ To check the 0-variation of a region $\eta_k(\alpha_{x,k})$, we need to compute $\text{diff}_n(\alpha_{x,n}, \alpha_{y,n})$ for many points $\alpha_{y,x}$ in $\eta_k(\alpha_{x,k})$, where $\text{diff}_n : D_{L_n} \times D_{L_n} \to \{0, 1\}$ is given by $\text{diff}_n(\alpha_{x,n}, \alpha_{y,n}) = 0$ if $\alpha_{x,n} = \alpha_{y,n}$ and 1 otherwise. Because $\alpha_{x,n}$ is known, we only need to compute $\alpha_{y,n}$. The tool computes $\alpha_{y,n}$ by finding a point $\alpha_{y,0} \in Pre_0(\alpha_{y,k})$ and then using the neural network to predict the value $\alpha_{y,n}$. It should be noted that all points in $Pre_0(\alpha_{y,k})$ have the same class, so any point in $Pre_0(\alpha_{y,k})$ is sufficient for our purpose.

To compute $\alpha_{y,0}$ from $\alpha_{y,k}$, we use functions $\psi_k, \psi_{k-1}, ..., \psi_1$ and compute points $\alpha_{y,k-1}, \alpha_{y,k-2}, ..., \alpha_{y,0}$ such that $\alpha_{y,j-1} = \psi_k(\alpha_{y,j})$ for $1 \leq j \leq k$. The computation relies on an SMT solver to encode the functions $\psi_k, \psi_{k-1}, ..., \psi_1$ if they are piecewise linear functions, and by taking the corresponding inverse functions directly if they are sigmoid functions.

D Input Parameters and Experimental Setup

The DLV tool accepts as input a network N and an image x, and has the following input parameters:

- an integer $l \in [0, n]$ indicating the starting layer L_l,
- an integer $dims_l \geq 1$ indicating the maximal number of dimensions that need to be considered on layer L_l,
- the values of variables s_p and m_p in V_l; for simplicity, we ask that, for all dimensions p that will be selected by the automated procedure, s_p and m_p have the same values,
- the precision $\varepsilon \in [0, \infty)$, and
- an integer $dims_{k,f}$ indicating the number of dimensions for each feature; for simplicity, we ask that every feature has the same number of dimensions and $dims_{k,f} = dims_{k',f}$ for all layers k and k'.

D.1 Two-Dimensional Point Classification Network

- $l = 0$
- $dims_l = 2$,
- $s_p = 1.0$ and $m_p = 1.0$,
- $\varepsilon = 0.1$, and
- $dims_{k,f} = 2$

D.2 Network for the MNIST Dataset

- $l = 1$
- $dims_l = 150$,
- $s_p = 1.0$ and $m_p = 1.0$,
- $\varepsilon = 1.0$, and
- $dims_{k,f} = 5$

D.3 Network for the CIFAR-10 Dataset

- $l = 1$
- $dims_l = 500$,
- $s_p = 1.0$ and $m_p = 1.0$,
- $\varepsilon = 1.0$, and
- $dims_{k,f} = 5$

D.4 Network for the ImageNet Dataset

- $l = 2$
- $dims_l = 20,000$,
- $s_p = 1.0$ and $m_p = 1.0$,
- $\varepsilon = 1.0$, and
- $dims_{k,f} = 5$