Robustness Evaluation of Deep Neural Networks with Provable Guarantees

Min Wu

Magdalen College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary Term 2020

To my parents,
Jingen Wu and Yuewen Wu.

In memory of my grandparents,
Pinsheng Wu and Meijin Zhu.

Acknowledgements

Personal

I would like to express my gratitude towards my supervisor, Professor Marta Kwiatkowska, without whose guidance, patience, and support this research would never have been completed. I also thank my Oxford collaborators for inspiring me to delve into the direction of verifying deep neural networks. I had such a great pleasure to work with them. Besides, to my friends at Oxford, thanks a lot for their kindness and tenderness. The past four years at Magdalen College shall be the summer of my life forever. Floreat Magdalena. Finally, I would like to thank my family for their everlasting love, especially my parents, Jingen and Yuewen, who have always supported and encouraged me to see the world.

Institutional

Some of the outcomes reported in this thesis are aligned with ongoing research projects led by my supervisor Marta Kwiatkowska, the EPSRC Programme Grant on Mobile Autonomy (EP/M019918/1) and FUN2MODEL (grant agreement No. 834115), funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme. Meanwhile, my doctoral research is funded by the CSC-PAG Oxford Scholarship (GAF1516_CSCUO_842704).

Abstract

This thesis presents methodologies to guarantee the *robustness* of deep neural networks, thus facilitating the deployment of deep learning techniques in safety-critical real-world systems. We study the *maximum safe radius* of a network with respect to an input, such that all the points within the radius are guaranteed to be safe, while, if exceeding the radius, there must exist an adversarial example. We extend the maximum safe radius to two variants: the *expected maximum safe radius* to evaluate global robustness of a dataset, and the *maximum safe radius w.r.t. optical flow* when the input is a video. We also study the *feature robustness* problem to quantify the robustness of features, extracted from an input, to adversarial perturbations. Specifically, we develop tensor-based parallelisation algorithms to compute the (expected) maximum safe radius of networks on (a set of) *pixel*-level images. For *features* of an image, we propose a game-based framework to compute the maximum safe radius and the feature robustness properties, where Player I selects features and Player II determines pixels within the feature to manipulate. Subsequently, we extend this game framework to *video* inputs to compute the maximum safe radius w.r.t. optical flow, with Player I choosing flows and Player II imposing modifications within the flow. As our work applies to large neural networks and high-dimensional inputs, we calculate the *upper and lower bounds* to approximate the maximum safe radius and the feature robustness, and guarantee that they converge to the optimal value, by utilising Lipschitz continuity. We implement the algorithms into three tools, DeepTRE, DeepGame, and DeepVideo, and demonstrate their effectiveness on benchmark image and video datasets.

Contents

List of Figures

1 Introduction

Researchers have always endeavoured to build "machines" to free humans from unnecessary waste of energy and time, from daily tedious tasks such as routine labour to more abstract problems that require complicated mathematical computation. For example, smart and complex software-based systems have been developed in various application domains of modern life, such as manufacturing, transportation, and healthcare. However, the ever-increasing sheer complexity of these machines makes it more and more painful and expensive for human designers to interpret their behaviours and specify all the relevant knowledge needed in the world.

To mitigate this conundrum, *artificial intelligence* (AI) [63] provides a feasible solution in the sense that the machines may become "intelligent" so that, through gathering knowledge from experience, they can learn and understand the world by themselves. Early successes of AI managed to efficiently solve problems that can be entirely described by a list of formal rules, in other words, that are straightforward for computers but intelligently challenging for humans. For example, the famous defeat achieved by IBM's Deep Blue in playing chess was a tremendous accomplishment, though, in retrospective, it was partially because the world of chess is not hugely complicated as all the movements are confined in a set of rigidly circumscribed ways.

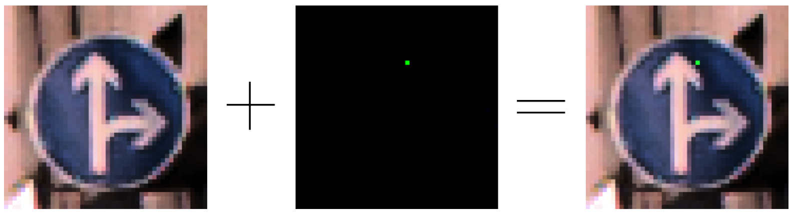


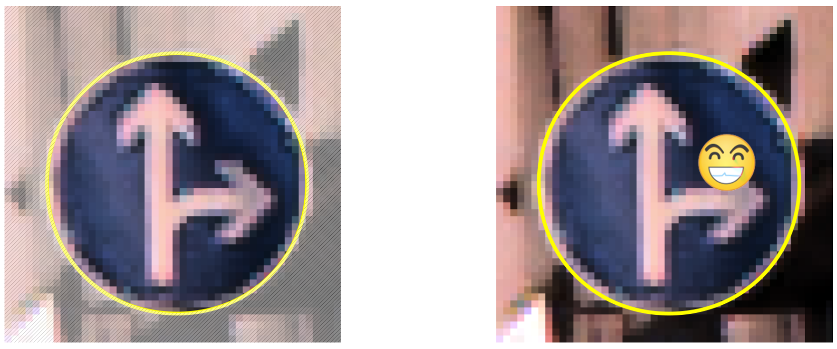
Figure 1.1: An *adversarial example* for a neural network trained on the GTSRB dataset. After a slight perturbation of just one pixel, i.e., Hamming distance $\eth_{\mathsf{Hamming}} = 1$, the image classification changes from "go **right** or straight" to "go **left** or straight".

Ironically, the real challenge of AI lies in the tasks that are actually easy for humans, that we can almost solve automatically and intuitively, such as understanding speech in daily conversations or recognising faces in photos. It is because, in our everyday life, there is an enormous amount of knowledge about the world, potentially subjective and difficult to articulate into formal descriptions, which the machines are incapable of capturing to behave intelligently.

A solution is *deep learning* [22, 36], which enables the machines to gather knowledge from experience and understand the world in respect of a hierarchy of concepts so that they can learn complex concepts via deriving them out of the simpler ones. For example, for a deep learning machine to understand the concept of the traffic sign in Figure 1.1, it can first gather the knowledge of edges, e.g., "the white circle", from the brightness discrepancy of the adjacent pixels, then build the concept of corners and contours based on the collections of the edges, and subsequently combine these corners and contours into specific object parts, e.g., "the two arrows", and finally recognise the identity of the object present in the image by taking into account all the object parts, i.e., "go right or straight". The quintessential model in deep learning is the *deep feed-forward network*, which is essentially a layer-by-layer composition of mathematical functions, regarding each function approximation of the outputs from the inputs as providing a new concept out of the simpler ones. As a specialised kind of feed-forward network, *convolutional networks* have been very successful in image classification, and as an extension with feedback connections, the *recurrent networks* have further powered natural language processing.

Despite the broad applications of deep learning in all sorts of real-world settings, sometimes reaching or even surpassing human-level performance, serious concerns have been raised regarding their security in safety-critical systems. In the context of autonomous driving, where neural network solutions have been proposed for tasks such as end-to-end steering, road segmentation, and traffic sign recognition, accidents such as the (fatal) crashes caused by Uber and Tesla self-driving cars in autopilot mode necessitate further research into the *robustness* of neural networks. One representative case is the discovery of *adversarial examples*. An adversarial example is a correctly-classified input which, after minor perturbations almost imperceptible to humans applied to it, can be misclassified by a network. For example, Figure 1.1 illustrates that, after modifying just a single pixel, say a tiny mud speckle or a water drop, the classification of the traffic sign surprisingly changes from "go right or straight" to "go left or straight", which may lead a car to steer off the road or even drive into oncoming traffic. Meanwhile, researchers in Berkeley found out that merely placing some "stickers" on a "stop" sign can easily fool a neural network into misclassifying it as "speed limit 45 mph". Though somewhat artificial, since in practice the self-driving car would rely on additional sensor inputs in the decision-making process, such cases strongly suggest that, before deployment in safety-critical tasks, neural networks' robustness to adversarial examples must be strengthened. Because such adversarial perturbations may take place in any of arbitrarily many dimensions, e.g., pixels or channels, of an image, we refer to it as the *robustness of neural networks on pixel-level images*.

Under some circumstances, however, instead of taking all input dimensions into consideration, it is more likely that we pay more attention to certain *features* of an input, partially due to the fact that they are more salient than others, or simply because it is not worth the resources to consider every single dimension. For instance, when trying to recognise the above-mentioned traffic sign, do human drivers actually care about the region outside of the highlighted yellow circle, as marked out in Figure 1.2(a)? The answer is apparently "No" as these pixels are irrelevant, and



(a) A highlighted feature (b) An emoji in the feature

Figure 1.2: *Feature* robustness of an image. For image classification networks, the actual traffic sign in the yellow circle should be more "robust" than the surrounding regions which are actually irrelevant when human drivers recognise "go right or straight". (a) A highlighted feature of an image. (b) A laughing emoji in the selected feature.

we almost immediately exclude them from our subconsciousness. Nevertheless, neural networks cannot intuitively make such a judgement. In experiments (such as Figure 4.3), we even noticed that some well-trained networks "see" the pixels in the background as more important than the main feature at the centre of an image. In other words, though beginning to reach human-level performance, they have not yet obtained a real human-level understanding. Ideally, the network should recognise the traffic sign in Figure 1.2(b) even with the presence of a laughing emoji in the highlighted feature, let alone in the surrounding regions. Regarding this, we evaluate the *robustness of neural networks on features of an image.*

To take a step further, in reality, our decision-making is more of a dynamic progress rather than a static one. That is, we perceive the surrounding environment constantly and then develop corresponding strategies. For example, in a driving scenario, human drivers continuously monitor nearby lane conditions while simultaneously taking control of the vehicle. Similarly as exhibited in Figure 1.3, for a self-driving car, it should "see" an approaching traffic sign, e.g., "speed limit 80 mph", getting closer and closer for a certain amount of time, then recognise the sign, and finally figure out the optimal strategy. In other words, the car evaluates

[1]The traffic sign recognition video link: `https://www.youtube.com/watch?v=XgOgbXMgF9w`

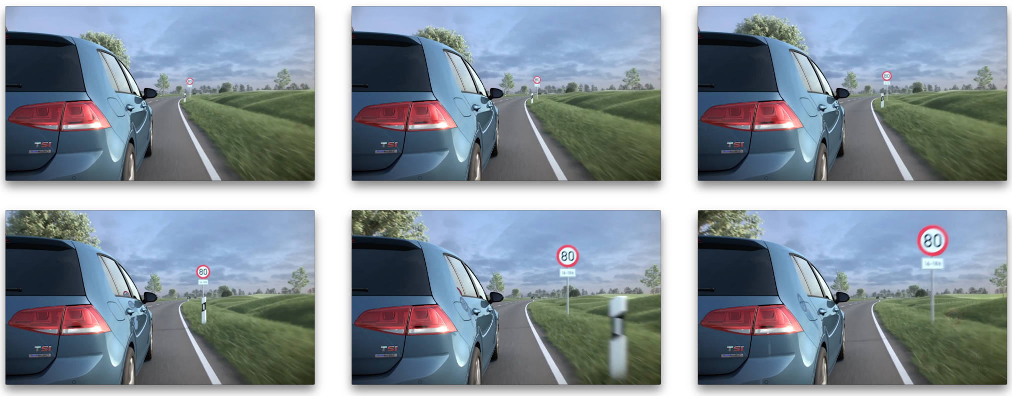


Figure 1.3: A *time-series* of images sampled from a traffic sign recognition video[1], in a top-down/left-right order. The self-driving car "sees" the road sign approaching for a certain duration, makes a judgement on the content on the sign, i.e., "speed limit 80 mph", and then reacts by mimicking human drivers.

a *time-series* of frames, i.e., a *video*, of the road condition possibly recorded by a camera instead of just a single image scan at some instant. To this end, we work on the *robustness of neural networks on videos*. This is an exciting problem as video classification is more challenging than image recognition because, apart from the spatial features on each frame, which can be extracted by the convolutional networks, videos also consist of the temporal dynamics between adjacent frames, where the recurrent networks come into place. Moreover, adversarial perturbations of videos include more variants than pixel manipulations in images, such as frame loss/repetition, brightness change, and camera occlusion.

To summarise, this thesis addresses the *robustness evaluation of deep neural networks*, specifically in a step-by-step manner, on pixel-level images, on features of an image, and on time-series of images, i.e., videos. Although the above motivation is largely described via traffic sign recognition in the context of autonomous driving, we contend that the proposed methodologies are also applicable to other image- or video-based scenarios, such as MRI (magnetic resonance imaging) scan in medical diagnosis and facial recognition in social media. Moreover, even beyond where the inputs can be represented as matrices/tensors, and where features can be extracted as subsets of the inputs, or where the inputs come sequentially, our work can be easily adapted to operate in complicated real-world environments, so as to establish confidence in the robustness of neural networks, and thereby facilitate their successful applications in society.

Contributions

As indicated in the title of the thesis, our work aims to provide *provable guarantees* for the robustness evaluation of deep neural networks. It is well known that deep learning algorithms tend to lack guarantees as the family of functions utilised is quite complicated. We tackle this from the perspective of *formal methods*, which are a particular set of mathematically based techniques, such as precise modelling and rigorous reasoning, for the specification and verification of sophisticated software and hardware systems, to ensure the safety, security, and robustness of these systems, especially in safety-critical applications.

To exploit formal methods in the context of deep learning, we develop algorithms and implement tools that can *verify* the correctness of the neural networks' behaviours concerning a given specification of a property. In other words, the specification has to hold for a specific input and all points in neighbourhood to the networks. For instance, given a property "for a self-driving car deployed with trained neural networks to monitor lane conditions, is it possible that it can always accurately recognise the traffic signs?", then in all possible situations involving the traffic signs, such as partial occlusion or weather change, we need to decide whether the networks can always classify correctly. The answer will be either it is `True`, or a *counter-example*, in this case, an *adversarial example* for which the violation occurs.

The main contributions regarding the *robustness evaluation of deep neural networks with provable guarantees* can be summarised in two aspects:

- We develop the theoretical foundations regarding the robustness evaluation of deep neural networks with provable guarantees. Explicitly, we define a distance, measured by metrics, called the *maximum safe radius* of a network with respect to an input, such that all the points within the distance are safe while, if exceeding the distance, there definitely exists an adversarial example. Based on this, we extend the maximum safe radius into two variants: (1) the *expected maximum safe radius* evaluating the global robustness of a whole dataset, and (2) the *maximum safe radius with respect to optical flow* when assessing the robustness guarantees for videos. Besides, we also propose the *feature robustness* problem. To compute the maximum safe radius and feature robustness, we exploit input space discretisation based on Lipschitz continuity, and thus reduce the robustness evaluation to *finite* optimisation problems. Since our work is applicable to large neural networks and real-world systems, in some situations, instead of computing the exact maximum safe radius or the feature robustness, we calculate the *upper and the lower bounds*, and prove that they converge to the optimal value. We also mention that, when evaluating the robustness with respect to a feature or an optical flow, we utilise a two-player turn-based *game* such that Player I selects a feature/flow and Player II determines atomic manipulations within the selected feature/flow.

- We demonstrate the effectiveness of our methodologies through implementing the algorithms into three tools, named DeepTRE for pixel-level robustness, DeepGame at the feature-level, and DeepVideo for videos, respectively, and analysing the experimental results. For all the tools, we perform robustness evaluation of neural networks on standard benchmark datasets, e.g., the MNIST, CIFAR-10, GTSRB, and ImageNet image datasets, as well as the UCF101 video dataset, and generate either the *exact* maximum safe radius or the *exact* feature robustness, or gradually and strictly *converging* upper and lower bounds. Besides, we remark that although designed to be verification tools, they can also perform competitive adversarial *attacks* after slight adaptation. Specifically, for each tool, DeepTRE can generate *saliency maps*, as a byproduct of computing the subspace sensitivity, which enables better interpretability and explainability of neural networks. Also, it utilises efficient *tensor-based parallelisation* so that during the computation procedure very few network queries are needed. Regarding DeepGame, it facilitates the analysis of networks' 'visual perception' in terms of identifying the robust features of an image. Also, various *feature extraction* methods, e.g., SURF or network logits, can be accommodated into the tool as long as an image partition can be achieved. In the game-based approximate verification framework, Monte Carlo tree search is applied to generate the upper bounds, and admissible A* and Alpha-Beta pruning are employed to produce the lower bounds for the maximum safe radius and the feature robustness computation, respectively. Finally, in DeepVideo we exploit the state-of-the-art VGG16 + LSTM network architecture to capture spatial and temporal features in terms of video robustness. Also, different optical flow methods are explored, such as the Horn-Schunck method and the Gunnar Farnebäck algorithm. In the game framework, a gradient-based search algorithm is used to generate the upper bounds, and admissible A* for the lower bounds.

Thesis Outline

This thesis is organised as follows. In Chapter 2 we review the related work in literature, and in Chapter 3 we introduce the technical background needed. The main contributions, i.e., the robustness evaluation of deep neural networks on pixel-level images, on features of an image, and on videos, are presented in Chapters 4, 5, and 6, respectively. We conclude with Chapter 7 summarising our work and highlighting possible future directions.

Publications

Some of the work included in this thesis has been previously published in papers with joint authors. Here, I clarify my contributions. To start with, the work concerning the global robustness evaluation of deep neural networks for the Hamming distance, which forms the basis of Chapter 4, first appeared in [60]. I contributed to the algorithm development and developed the tool DeepTRE in Python, and produced the experimental results regarding adversarial attacks on the benchmark image datasets, MNIST and CIFAR-10, in comparison to some existing tools, such as JSMA and C&W, in terms of the Hamming distance and computational time. As a by-product, I also generated some of the ImageNet saliency maps. Besides, since our methodology takes advantage of tensor-based parallelisation, I also carried out the comparative experiments using the Nvidia GPUs and the CPUs in my departmental desktop. As for the robustness evaluation of deep neural networks on feature-level images, presented in Chapter 5, the game-based approximate verification framework with converging upper and lower bounds is published in [80]. Apart from collaborating with the co-authors on developing the concept and algorithms, I contributed to the tool DeepGame the implementation of the lower bounds in both the cooperative and the competitive games, utilising Admissible A* and Alpha-Beta Pruning, respectively. Moreover, I produced all the experimental results that were generated by this tool, including the convergence analysis of the bounds on MNIST, CIFAR-10, and GTSRB, as well as the comparison of various tools on adversarial attacks.

Meanwhile, some of the work in this thesis is in preparation for publication and can be found online. For instance, the contents of the robustness guarantees for deep neural networks on videos in Chapter 6 are primarily from [78], which I co-authored with my supervisor. Also, I wrote part of the review on the verification approaches for deep neural networks in [27], which forms the basis of Chapter 2.

Apart from these, though not included in this thesis, I also contributed to some other publications, such as [79], [28], and [67].

2 Related Work

Contents

In this chapter, we review some work related to the robustness evaluation of deep neural networks. They are introduced from two perspectives: in Section 2.1, the *verification* approaches of neural networks in the community of formal methods, and in Section 2.2, the *adversarial attacks* from the computer vision and security communities. The key difference is that the former provides guarantees whilst the latter does not. Instead, the attacking methods check whether a network can be easily attacked to see the network's possible lack of robustness. As our work can provide provable guarantees, in Section 2.3, we compare with those verification approaches only, though those adversarial attacks mentioned are used as the baseline methods in Chapters 4 and 5 to perform comparison in generating adversarial examples.

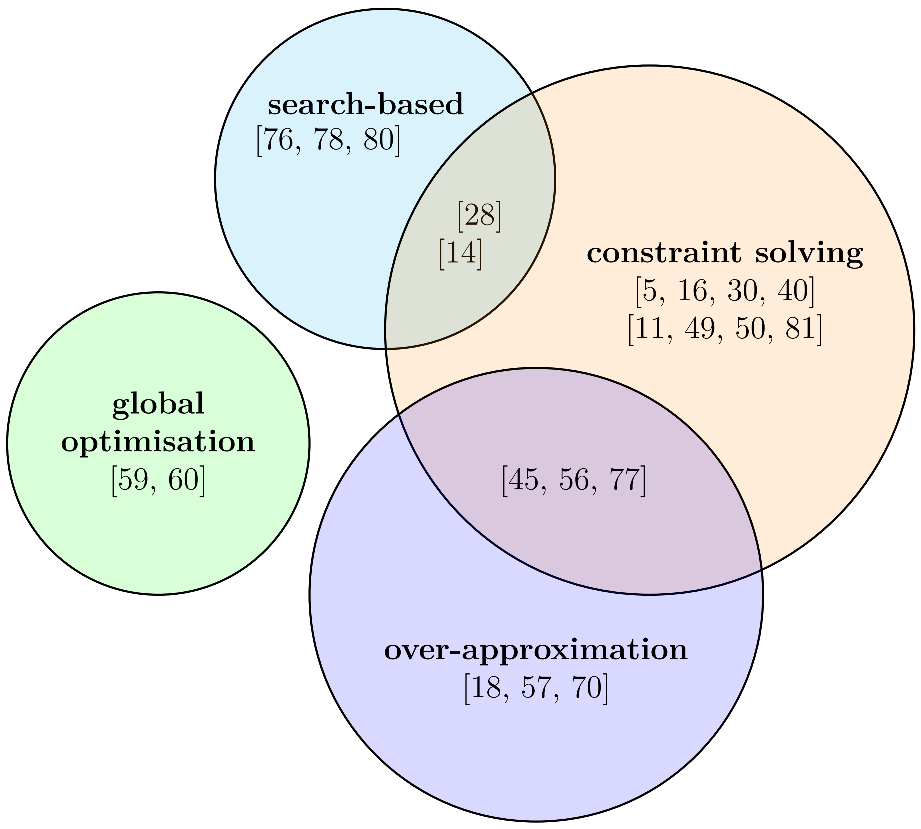


Figure 2.1: A (not rigid) taxonomy of verification works for (feed-forward) neural networks. The overlapping areas indicate that the approaches fall into both categories.

2.1 Verification of Deep Neural Networks

According to the underlying techniques, existing work on verification of deep neural networks largely fall into the following categories: constraint solving, search-based approach, global optimisation, and over-approximation, although the separation between them may not be strict. Figure 2.1 illustrates a taxonomy of the methods mentioned in this chapter. A few survey papers that provide a broader overview of the algorithms for verifying neural networks are available, such as [39] and [27].

In this chapter, we take the approach in [27] by classifying verification techniques with respect to the type of guarantees they can provide. Basically, the guarantees can be:

- *Exact deterministic* guarantee, which states exactly whether a property holds.

We will omit the word *exact* and call it deterministic guarantee for simplicity.

- *One-sided* guarantee, which provides either a lower bound or an upper bound for a variable, and thus can serve as a sufficient condition for a property to hold - the variable can denote, e.g., the greatest value of some dimension in the output reachable set.
- Guarantee with *converging* lower and upper bounds to a variable.
- *Statistical* guarantee, which quantifies the probability that a property holds.

Apart from the category of achievable guarantees, we also mention some of the objective properties: the *local robustness* property, which requires a network's invariance against small perturbations; the *output reachability* property, which computes a set of outputs given a set of inputs; the *interval* property, which computes a convex over-approximation of the output reachable set, and finally the *Lipschitzian* property, which monitors the changes of the outputs with respect to the small changes of the inputs. Detailed definitions of these safety properties can be found in Section 3.3.

2.1.1 Approaches with Deterministic Guarantees

Deterministic guarantees are achieved by transforming a verification problem into a set of constraints (with or without optimisation objectives), so that they can be solved with a constraint solver. The name "deterministic" comes from the fact that solvers usually return a deterministic answer to a query, i.e., either satisfiable or unsatisfiable. This is based on the current success of various constraint solvers such as SAT solvers, linear programming (LP) solvers, mixed integer linear programming (MILP) solvers, and Satisfiability Modulo Theories (SMT) solvers.

SMT/SAT

An abstraction-refinement approach based on SMT solving. A solution to the verification of the interval property (which can be easily extended to work with the reachability property for ReLU activation functions) is proposed in [56] by abstracting a DNN into a set of Boolean combinations of linear arithmetic constraints. It is shown that, whenever the abstracted model is declared to be safe, the same holds for the concrete one. Spurious counter-examples, on the other hand, trigger refinements and can be leveraged to automate the correction of misbehaviour. This approach is validated on DNNs with fewer than 10 neurons, with logistic activation function.

SMT solvers for DNNs. Two SMT tools, Reluplex [30] and Planet [16], were put forward to verify DNNs on properties expressible with SMT constraints. SMT solvers often have good performance on problems that can be represented as a Boolean combination of constraints over other variable types. Typically, an SMT solver combines a SAT solver with specialised decision procedures for other theories. In the verification of DNNs, they adapt linear arithmetic over real numbers, in which an atom (i.e., the most basic expression) is of the form $\sum_{i=1}^{n} a_i x_i \leq b$, where a_i and b are real numbers.

In both Reluplex and Planet, they rely on the classical Davis-Putnam-Logemann-Loveland (DPLL) algorithm in splitting cases and ruling out conflict clauses, while they differ slightly in dealing with the intersection. For Reluplex, it inherits rules in the algorithm of Simplex and adds some non-linear rules for the ReLU operation. Through the classical pivot operation, it first looks for a solution for the linear constraints, and then applies the rules for ReLU to satisfy the ReLU relation for every node. Differently, Planet uses linear approximation to over-approximate the neural network, and manages the condition of ReLU and max-pooling nodes with logic formulas.

SAT approach. [49, 50] propose to verify properties of a class of neural networks, i.e., binarised neural networks, in which both weights and activations are binary, by reduction to the well-known Boolean satisfiability. Using this Boolean encoding, they leverage the power of modern SAT solvers along with a proposed counterexample-guided search procedure to verify various properties of these networks. A particular focus is on the robustness to adversarial perturbations. The experimental results demonstrate that this approach scales to medium-size deep neural networks used in image classification tasks.

Mixed Integer Linear Programming (MILP)

MILP formulation for DNNs. [40] encodes the behaviours of fully connected neural networks with MILP. For instance, a hidden layer $z_{i+1} = \mathrm{ReLU}(W_i z_i + b_i)$ can be described with the following MILP:

$$\begin{aligned}
&z_{i+1} \geq W_i z_i + b_i, \\
&z_{i+1} \leq W_i z_i + b_i + M t_{i+1}, \\
&z_{i+1} \geq 0, \\
&z_{i+1} \leq M(1 - t_{i+1}),
\end{aligned}$$

where t_{i+1} has value 0 or 1 in its entries and has the same dimension as z_{i+1}, and $M > 0$ is a large constant which can be treated as ∞. Here each integer variable in t_{i+1} expresses the possibility that a neuron is activated or not. The optimisation objective can be used to express properties related to the bounds, and therefore this approach can work with both reachability and interval properties.

However, it is not efficient to simply use MILP to verify DNNs or to compute the output range. In [11], a number of MILP encoding heuristics are developed to speed up the solving process, and moreover parallelisation of MILP-solvers is used to result in an almost linear speed-up in the number (up to a certain limit) of computing cores in experiments. In [14], Sherlock alternately conducts local and global search to efficiently calculate the output range. In a local search phase, it uses gradient descent method to find a local maximum (or minimum), while in a global search phase, it encodes the problem with MILP to check whether the local maximum (or minimum) is in the global output range.

Moreover, [5] presents a branch and bound (B&B) algorithm and claims that both SAT/SMT-based approaches and MILP-based approaches can be regarded as its special cases.

2.1.2 Approaches to Compute an Approximate Bound

The approaches to be surveyed in this subsection consider the computation of a *lower* (or by duality, an *upper*) bound, and are able to claim the sufficiency of achieving given properties. While these approaches can only have a bounded estimation to the value of some variable, they are able to work with larger models, e.g., up to 10 000 hidden neurons. The other advantage is the potential to avoid floating point issues in existing constraint solver implementations. Actually, most state-of-the-art constraint solvers implementing floating-point arithmetic only give approximate solutions, which may not be the actual optimal solution or may even lie outside the feasible space [51]. It may happen that a solver wrongly claims the satisfiability or un-satisfiability of a property. For example, [14] reports several false positive results in Reluplex, and mentions that this may come from unsound floating point implementation.

Abstract Interpretation

Abstract interpretation is a theory of sound approximation of the semantics of computer programs [12]. It has been used in static analysis to verify properties of a program without actually running it. The basic idea of abstract interpretation is to use abstract domains (represented as, e.g., boxes, zonotopes, polyhedra) to over-approximate the computation of a set of inputs. It has been explored in a few papers, including [18, 38, 45].

Generally, on the input layer, a concrete domain $\mathcal{C}$ is defined such that the set of inputs η is one of its elements. To enable efficient computation, a comparatively simple domain, i.e., abstract domain $\mathcal{A}$, which over-approximates the range and relation of variables in $\mathcal{C}$, is chosen. There is a partial order $\leq$ on $\mathcal{C}$ as well as $\mathcal{A}$, which is the subset relation $\subseteq$.

Definition 2.1 (Galois Connection)**.** *A pair of functions $\alpha : \mathcal{C} \to \mathcal{A}$ and $\gamma : \mathcal{A} \to \mathcal{C}$ is a* Galois connection *if, for any $a \in \mathcal{A}$ and $c \in \mathcal{C}$, we have $\alpha(c) \leq a \Leftrightarrow c \leq \gamma(a)$.*

Intuitively, a Galois connection (α, γ) expresses abstraction and concretisation relations between domains, respectively. A Galois connection is chosen because it preserves the order of elements in two domains. Note that $a \in \mathcal{A}$ is a sound abstraction of $c \in \mathcal{C}$ if and only if $\alpha(c) \leq a$.

In abstract interpretation, it is important to choose a suitable abstract domain because it determines the efficiency and precision of the abstract interpretation. In practice, a certain type of special shapes is used as the abstraction elements. Formally, an abstract domain consists of shapes expressible as a set of logical constraints. The most popular abstract domains for the Euclidean space abstraction include Interval, Zonotope, and Polyhedron.

The approaches based on abstract interpretation can verify interval properties, but cannot verify reachability properties.

Convex Optimisation

A method is proposed in [77] to learn deep ReLU-based classifiers that are provably robust against norm-bounded adversarial perturbations on the training data. The approach works with the interval property, but not the reachability property. It may flag some non-adversarial examples as adversarial examples. The basic idea is to consider a convex outer over-approximation of the set of activations reachable through a norm-bounded perturbation, and then develop a robust optimisation procedure that minimises the worst-case loss over this outer region (via a linear program). Crucially, it is shown that the dual problem to this linear program can be represented itself as a deep network similar to the back-propagation network, leading to very efficient optimisation approaches that produce guaranteed bounds on the robust loss. The approach is illustrated on a number of tasks with robust adversarial guarantees. For example, for MNIST, they produce a convolutional classifier that provably has less than 5.8% test error for any adversarial attack with bounded Chebyshev distance (L^∞ norm) less than $\epsilon = 0.1$.

Moreover, [15] works by taking a different formulation of the dual problem, i.e., applying Lagrangian relaxation on the optimisation. This is to avoid working with constrained non-convex optimisation problem.

Interval Analysis

In [70], the interval arithmetic is leveraged to compute rigorous bounds on the DNN outputs, i.e., the interval property. The key idea is that, given the ranges of operands, an over-estimated range of the output can be computed by using only the lower and upper bounds of the operands. Starting from the first hidden layer, this computation can be conducted through to the output layer. Beyond this explicit computation, symbolic interval analysis along with several other optimisations are also developed to minimise over-estimations of output bounds. These methods are implemented in ReluVal, a system for formally checking security properties of ReLU-based DNNs. An advantage of this approach, compared to constraint-solving based approaches, is that it can be easily parallelised. In general, interval analysis is close to the interval-based abstract interpretation (Section 2.1.2).

In [55], lower bounds of adversarial perturbations needed to alter the classification of the neural networks are derived by utilising the layer functions. The proposed bounds have the theoretical guarantee that no adversarial manipulation could be any smaller, and in this case can be computed efficiently - at most linear time in the number of (hyper)parameters of a given model and any input, which makes them applicable for choosing classifiers based on robustness.

Output Reachable Set Estimation

In [81], the output reachable set estimation is addressed. Given a DNN $\mathcal{N}$ with its associated function f, and a set of inputs η, the output reachable set is $\textsc{Reach}(f, \eta)$ as in Definition 3.10. The problem is to either compute a close estimation $\mathcal{Y}'$ such that $\textsc{Reach}(f, \eta) \subseteq \mathcal{Y}'$, or to determine whether $\textsc{Reach}(f, \eta) \cap \neg\mathcal{S} = \emptyset$ for a safety specification $\mathcal{S}$, where $\mathcal{S}$ is also expressed with a set similar to that in Equation (3.26). Therefore, it is actually to compute the interval property. First, a concept called maximum sensitivity is introduced and, for a class of multi-layer perceptrons whose activation functions are monotonic functions, the maximum sensitivity can be computed via solving convex optimisation problems. Then, using a simulation-based method, the output reachable set estimation problem for neural networks is formulated into a chain of optimisation problems. Finally, an automated safety verification is developed based on the output reachable set estimation result. The approach is applied to the safety verification for a robotic arm model with two joints.

Linear Approximation of ReLU Networks

FastLin/FastLip [74] analyses ReLU networks on both the interval property and the Lipschitzian property. For the interval property, they consider linear approximation over those ReLU neurons that are uncertain on their status of being activated or deactivated. For the Lipschitzian property, they use the gradient computation for the approximate computation. Crown [82] generalises the interval property computation algorithm in [74] by allowing the linear expressions for the upper and lower bounds to be different and enabling its working with other activation functions such as tanh, sigmoid and arctan. The Lipschitzian property computation is improved in RecurJac [83].

2.1.3 Approaches to Compute Converging Bounds

While the above approaches can work with small networks (up to a few thousands of hidden neurons), state-of-the-art DNNs usually contain at least many-million hidden neurons. It is necessary to develop approaches to work with real-world systems. In Sections 2.1.3 and 2.1.4, we discuss approaches able to work with large-scale networks, although they might have other restrictions or limitations. Since the approaches surveyed in this subsection compute converging upper and lower bounds, they can work with both output reachability and interval properties.

Layer-by-Layer Refinement

DLV [28] develops an automated verification framework for feedforward multi-layer neural networks based on Satisfiability Modulo Theory (SMT). The key features of this framework are that it *guarantees* a misclassification being found if it exists, and that it propagates the analysis *layer-by-layer*, i.e., from the input layer to, in particular, the hidden layers, and to the output layer.

In this work, *safety* for an individual classification decision, i.e., pointwise (or local) robustness, is defined as the invariance of a classifier's outcome to perturbations within a small neighbourhood of an original input. Formally,

$$\mathcal{N}, \eta_k, \delta_k \models x$$

where x denotes an input, $\mathcal{N}$ a neural network, η a region surrounding the input, δ a set of manipulations, and subscript k means at layer k. Later, in [76, 80], it is shown that the minimality of the manipulations in δ can be guaranteed with the existence of a Lipschitz constant.

To be more specific, the verification algorithm uses single-/multi-path search to exhaustively explore a finite region of the vector spaces associated with the input layer or the hidden layers, and a layer-by-layer refinement is implemented using the Z3 solver to ensure that the local robustness of a deeper layer implies the robustness of a shallower layer. The methodology is implemented in the software tool DLV, and evaluated on image benchmarks such as MNIST, CIFAR-10, GTSRB, and ImageNet. Though the complexity is high, it scales to work with state-of-the-art networks such as VGG16. Furthermore, in [76, 80], the search problem is alleviated by Monte Carlo tree search.

Global Optimisation Based Approaches

DeepGO [59] shows that most known layers of DNNs are Lipschitz continuous, and presents a verification approach based on global optimisation. For a single dimension, an algorithm is presented to always compute the lower bounds (by utilising the Lipschitz constant) and eventually converge to the optimal value. Based on this single-dimensional algorithm, the algorithm for multiple dimensions is to exhaustively search for the best combinations. The algorithm is able to work with state-of-the-art DNNs, but is restricted by the number of dimensions to be perturbed.

2.1.4 Approaches with Statistical Guarantees

This subsection reviews a few approaches aiming to achieve statistical guarantees on their results, by claiming, e.g., the satisfiability of a property, or a value is a lower bound of another value, etc., with certain probability.

Lipschitz Constant Estimation by Extreme Value Theory

[75] proposes a metric, called CLEVER, to estimate the Lipschitz constant, i.e., the approach works with the Lipschitzian property. It estimates the robustness lower bound by sampling the norm of gradients and fitting a limit distribution using extreme value theory. However, as argued by [20], their evaluation approach can only find statistical approximation of the lower bound, i.e., their approach has a soundness problem.

Robustness Estimation

The work [1] proposes two statistics of robustness to measure the frequency and the severity of adversarial examples, respectively. Both statistics are based on a parameter ϵ, which is the maximum radius within which no adversarial examples exist. The computation of these statistics is based on the local linearity assumption which holds when ϵ is small enough. Except for the application of the ReLU activation function which is piece-wise linear, this assumption can be satisfied by the existence of the Lipschitz constant as shown in [59].

2.1.5 Computational Complexity of Verification

There are two main ways to measure the complexity of conducting formal verification. The first, which appeared in [30], measures the complexity with respect to the number of hidden neurons. This is due to the fact that their approach is to encode the DNN into a set of constraints, and in the constraints every hidden neuron is associated with two variables. On the other hand, in [59], the complexity is measured with respect to the number of input dimensions. This is due to the fact that their approach is to manipulate the input. For both cases, the complexity is shown to be NP-complete, although it is the case that the number of hidden neurons can be larger than the number of input dimensions.

2.2 Adversarial Attacks

Given an input, an *adversarial attack* (or *attacker*) attempts to craft a perturbation or distortion to the input to make it misclassified, often with high confidence, by a well-trained deep neural network. Attack techniques can roughly be divided into two groups depending on the type of misclassification objective:

- For a *targeted* perturbation, the attacker can control the resulting misclassification label.
- For an *un-targeted* perturbation, the attacker can search for a misclassification but cannot control its resulting misclassification label.

Meanwhile, based on the amount of information an attacker can access, adversarial perturbations can also be organised into two categories:

- *White-box* perturbation – the attacker needs to access the parameters and the internal structure of the trained networks, and may also need to access the training dataset.

- *Black-box* perturbation – an attacker can only query the trained networks with perturbed inputs, without the ability to access the internal structure and parameters.

Moreover, according to the distance metrics adopted to evaluate the discrepancy between a perturbed input and the original input, adversarial attacks can be listed as the Hamming distance, the Manhattan distance (L^1 norm), the Euclidean distance (L^2 norm) or the Chebyshev distance (L^∞ norm) attacks. Note that, while any of these metrics can measure all perturbations, an attack technique can sometimes generate adversarial examples that are better measured by a particular metric.

In the computer vision community, various attack techniques to generate such adversarial examples have recently been developed. From a technical point of view, such attacks can be categorised as using cost gradient [4, 21, 46] or the forward gradient of the networks [53], or perturbing towards the most promising directions, or directly solving an optimisation problem (possibly using gradient ascent/descent) to obtain a perturbation [8, 47]. Interestingly, adversarial examples have revealed to be transferable between different network architectures, or between networks trained on disjoint subsets of data [53, 68], or even between real-world scenarios [35]. For example, adversarial images remain misclassified even after being printed out and recaptured with a cell phone camera. In the following, we mention a few adversarial attacks that are most relevant to this thesis.

Limited-Memory BFGS Algorithm (L-BFGS)

Szegedy et al. [68] noticed the existence of adversarial examples in the neighbourhood of the correctly-classified inputs and described them as the "blind spots" of neural networks. The authors claim that, because the adversarial examples have a low probability of occurrence, they cannot be found efficiently via sampling around the correctly-classified inputs, yet can be discovered through an optimisation scheme. Formally, assume we have a classifier $f : \mathbb{R}^m \mapsto C = \{1, \ldots, n\}$ that maps each input to a class $c \in C$, a given input $\boldsymbol{x} \in \mathbb{R}^m$, and a target label $c' \in C$ such that $c' \neq \arg\max f(\boldsymbol{x})$, the goal is to find an additive adversarial perturbation $\boldsymbol{r} \in \mathbb{R}^m$ with the following optimisation expression:

- Minimise $\|\boldsymbol{r}\|_2$ subject to:
    1. $\arg\max f(\boldsymbol{x}+\boldsymbol{r}) = c'$, and
    2. $\boldsymbol{x}+\boldsymbol{r} \in \mathbb{R}^m$.

As exact computation is hard, an approximate algorithm based on the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm is proposed. Furthermore, [68] discovers that adversarial examples are abundant, and, utilising the above framework, one could generate an unlimited number of them. The authors also remark that these adversarial examples generalise across both model and training sets – an adversarial example generated for one network classifier will likely be an adversarial example for another classifier with different architectures or training datasets. Adversarial examples were also independently discovered by [23] in the context of security.

Fast Gradient Sign Method (FGSM)

The Fast Gradient Sign Method (FGSM) [21] can find adversarial perturbations with a fixed L^∞ norm constraint, in the sense that it conducts a one-step modification to all pixel values so that the value of the loss function increases under the L^∞ norm constraint. The authors argue that this provides a linear explanation of the existence of adversarial examples. They highlight that, since the precision of an individual input feature tends to be limited, e.g., digital images often use only 8 bits per pixel and therefore are precise up to $^1/_{255}$, it is unreasonable for a classifier to respond differently to two inputs if they only differ on each feature by an amount that is less than the level of precision. However, consider the dot product between a weight vector $\boldsymbol{W}$ and an adversarial example $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{r}$, i.e.,

$$\boldsymbol{W}^T\boldsymbol{x}' = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{W}^T\boldsymbol{r},$$

and let $\boldsymbol{r} = \epsilon\ \mathrm{sign}(\boldsymbol{W})$, then the activation growth can be maximised this way. If $\boldsymbol{W}$ has m dimensions with elements having average magnitude n, the activation growth is ϵmn, i.e., increases linearly with respect to the dimensionality of the problem, whereas $\|\eta\|_\infty$ remains less than ϵ. Thus, for high-dimensional problems, FGSM can make many small changes to the input to produce a significant difference in model output. Based on this linear explanation, [21] suggests a fast linear algorithm to generate adversarial examples. Denoting θ as the model parameters, $\boldsymbol{x}$ an input to the model, c the label associated with $\boldsymbol{x}$, and $J(\theta, \boldsymbol{x}, c)$ the cost function used to train the model, an adversarial perturbation $\boldsymbol{r}$ can be generated by

$$\boldsymbol{r} = \epsilon\ \mathrm{sign}\left(\nabla_{\boldsymbol{x}} J(\theta, \boldsymbol{x}, c)\right). \tag{2.1}$$

A greater ϵ leads to a higher success rate of attack but potentially results in a more significant human perception contrast. Later, this attack method is extended to a targeted and iterative version [35].

Jacobian Saliency Map based Attack (JSMA)

Papernot et al. [53] present the Jacobian Saliency Map based Attack (JSMA) algorithm based on the *forward derivative* of a network, which is defined as the Jacobian matrix of the output probability distribution, over the set C of labels, with respect to the input dimensions. The forward derivative highlights those features that the network's prediction is most sensitive to and are thus most likely to cause misclassification when disturbed. For a given target class $c \in C$ and a given input $\boldsymbol{x} \in [0, 1]^m$, each dimension of the input is assigned a salient value based on the forward derivative. The salient value captures, for each input dimension, the sensitivity of the output probability assigned to a class c. For the adversarial perturbation, the input dimension with the highest salient value is perturbed by a *maximum distortion parameter* $\tau > 0$. If this perturbation results in a misclassification, then the algorithm terminates; otherwise, the forward derivative is computed again over the distorted input and the algorithm proceeds.

The algorithm may also terminate when a maximum distance threshold $d > 0$ is reached. This algorithm does not require the computation of the derivative of the perturbation measured with the L^p norm, and can be used to generate adversarial perturbations that are minimised under the Hamming distance. This method is generally slower than FGSM and aims to find an adversarial example that has a lower Hamming distance to the legitimate image.

Carlini & Wagner Attack (C&W)

The C&W attack [8] is an optimisation-based attack method which formulates finding an adversarial example as an image distance minimisation problem for metrics such as the Hamming distance, the Euclidean distance (L^2 norm), and the Chebyshev distance (L^∞ norm). Specifically, it defines an optimisation problem over the loss function

$$\texttt{loss}(\boldsymbol{r}) = \|\boldsymbol{r}\|_p + c \cdot f(\boldsymbol{x} + \boldsymbol{r}), \tag{2.2}$$

where f is a function that is negative when the network $\mathcal{N}$ misclassifies $\boldsymbol{x} + \boldsymbol{r}$, under the constraint that $\boldsymbol{x} + \boldsymbol{r}$ is a valid input. The optimisation problem is solved using the Adam gradient-descent method described in [31]. This approach is applied for three distance metrics, and the algorithm is adjusted slightly in each case. In particular, for the Hamming distance, an iterative algorithm identifies a subset of features having low impact on classification that are thus not considered candidates for perturbation. This subset grows with each iteration, until its complement set is sufficiently small, giving a minimal feature subset salient to classification. In each iteration the feature i selected for exclusion is the one that minimises $\nabla f(\boldsymbol{x}+\boldsymbol{r})_i \cdot \boldsymbol{r}_i$. A smart trick in C&W attacker lies on introducing a new optimisation variable to avoid box constraint (image pixels need to be within $[0, 1]$). This approach is similar in intuition to [53], in that a subset of salient features is identified based on the first-order derivatives, and [8] is shown to be more effective than [53]. C&W attack can find an adversarial example that has a significantly smaller image distance, especially based on the L^2 norm.

2.3 Summary

To summarise, we identify the limitations present in the literature on verification of deep neural networks, and show that our approaches have addressed the gap.

- To start with, most existing methods evaluate robustness based on the L^p, $p \geq 1$, norms, which are differentiable, so that most optimisation algorithms such as gradient descent can be utilised. In Chapter 4, we focus on the non-differentiable Hamming distance and propose a technique, which, as far as we know, is the *first* time to guarantee network safety on this specific metric.
- Moreover, while often a single input as a whole is evaluated against adversarial perturbations, we extend this to the features of an input in Chapter 5, and therefore to facilitate the explainability and interpretability of networks in terms of how they actually 'see' and/or 'understand' an input. For instance, do they focus on the salient feature, or the set of salient features, in a manner consistent with human perception? To the best of our knowledge, ours is the *first* method that guarantees feature-level robustness of neural networks.
- To take a step further, we generalise the input to have time-series characteristics, as human decision-making processes tend to be dynamic and continuous. In particular, we study the times-series of images, i.e., videos, which capture spatial features on individual frames and temporal features between adjacent frames. Correspondingly, robustness is guaranteed for networks with both convolutional and recurrent layers. Before the work in Chapter 6 was put online, *no other* work in public domain addressed robustness guarantees for videos.

We summarise the capabilities of existing approaches to the verification of deep neural networks discussed here, also including our approaches, in Table 2.1, taking into account the type of achievable guarantees, underlying algorithms, and objective properties, i.e., robustness, reachability, interval, and Lipschitzian. We remark that our work is scalable to large neural networks and applicable to real-world scenarios. Moreover, because computing the precise deterministic guarantees is unrealistic for high dimensional inputs and networks, we provide anytime algorithms for upper and lower bounds that converge to the optimal value. In other words, our approaches mostly fall into the category of Section 2.1.3, where converging bounds are computed.

Table 2.1: Comparison between the verification approaches of deep neural networks. Our methods are highlighted in bold, such as the pixel-level robustness [60] in Chapter 4, the feature-level robustness [80] in Chapter 5, and the video-level robustness [78] in Chapter 6.

	Guarantees	**Algorithm**		**Property**			
				Robustness	*Reachability*	*Interval*	*Lipschitzian*
[56]	Deterministic Guarantees	Constraints Solving	SMT	✓	✓	✓	
[30]				✓	✓	✓	
[16]				✓	✓	✓	
[50]			SAT	✓	✓	✓	
[40]		MILP		✓	✓	✓	
[11]				✓	✓	✓	
[14]				✓	✓	✓	
[5]				✓	✓	✓	
[18]	Lower/Upper Bound	Abstract Interpretation		✓		✓	
[45]				✓		✓	
[38]				✓		✓	
[77]		Convex Optimisation		✓		✓	
[70]		Interval Analysis		✓		✓	
[55]				✓		✓	
[81]		Set Estimation		✓		✓	
[74]		Linear Approximation		✓		✓	✓
[28]	Converging Bounds	Search Based	Layer-by-Layer Refinement	✓	✓	✓	
[76]			Two-Player Turn-based Game	✓	✓	✓	✓
[80]				✓	✓	✓	✓
[78]				✓	✓	✓	✓
[59]		Global Optisimation		✓	✓	✓	✓
[60]				✓	✓	✓	
[75]	Statistical Guarantees	Extreme Value Theory					✓
[1]		Robustness Estimation		✓			

3 Background

Contents

In this chapter, we introduce the technical background materials. Starting with Section 3.1, we present the definitions of the three categories of deep neural networks used in the thesis, i.e., the feed-forward neural networks, the convolutional neural networks, and the recurrent neural networks. Then, after explaining the distance metrics and Lipschitz continuity in Section 3.2, we discuss four safety properties of neural networks in Section 3.3. In Section 3.4, we describe the techniques that are related to the inputs, i.e, the images and the videos, such as the tensor folding and unfolding operations, feature extraction for images, and the optical flow methods for videos.

3.1 Deep Neural Networks

3.1.1 Feed-Forward Neural Networks

Feed-forward neural networks, also called *deep feed-forward networks*, or *multilayer perceptrons* (MLPs), are the quintessential deep learning models. The goal of a feed-forward neural network is to approximate some function, for instance, to imitate human recognition in image classification tasks. These networks are called *neural* because they are loosely inspired by neuroscience, and are typically represented by composing different functions together, thus *networks*. Here, *feed-forward* indicates that the information evaluated from the inputs flows through the intermediate computations to the outputs, and that there are no feedback connections where outputs of the network are fed back to itself. In fact, if there are feedback connections included, they are called *recurrent neural networks*, as presented in Section 3.1.3.

Definition 3.1 (Feed-Forward Neural Networks). *A* feed-forward neural network *is a weighted, directed, acyclic graph $\mathcal{N} = (N, V, w, b)$, where*

- *N is a set of* neurons*;*
- *V is a set of* connections *between neuron pairs $(n_i, n_j), n_i, n_j \in N$;*
- *$w : V \to \mathbb{R}$ defines the* weight *of a connection;*
- *$b : N \to \mathbb{R}$ defines the* bias *of a neuron;*

and the following properties hold:

- *A* layer *$L^{(k)} \subseteq N, k \in \{1, 2, \ldots, l\}$ is an ordered, mutually exclusive set* *of neurons, such that*

$$N = \bigcup_{k=1}^{l} L^{(k)}, \quad \textit{for} \quad l \in \mathbb{N}^{+} \tag{3.1}$$

where l is the number of layers. Moreover, for each neuron pair (n_i, n_j) there exists a unique k such that $n_i \in L^{(k)}$ and $n_j \in L^{(k+1)}$.

- *An* activation function $f^{(k)} : \mathbb{R} \to \mathbb{R}$ *exists for each layer $L^{(k)}$.*

Weights and Biases Consider a feed-forward neural network $\mathcal{N}$ and its k-th layer $L^{(k)}$, then the *weight matrix* $\boldsymbol{W}^{(k)} \in \mathbb{R}^{m \times n}$, where $m = \left|L^{(k-1)}\right|$ and $n = \left|L^{(k)}\right|$, is a matrix with elements defined from the weight function: $w_{ij}^{(k)} = w(L_i^{(k-1)}, L_j^{(k)})$; and the *bias vector* $\boldsymbol{b}^{(k)} \in \mathbb{R}^n$ is a vector with elements defined from the bias function: $b_j^{(k)} = b(L_j^{(k)})$.

Computation by Neural Networks Using the weights and biases defined above, for a specific layer $L^{(k)}$ the activation function $f^{(k)} : \mathbb{R}^m \to \mathbb{R}^n$, where $m = \left|L^{(k-1)}\right|$ and $n = \left|L^{(k)}\right|$, is given by

$$f^{(k)}(\boldsymbol{x}) \stackrel{\text{def}}{=} f^{(k)}(\boldsymbol{W}^{(k)}\boldsymbol{x} + \boldsymbol{b}^{(k)}). \tag{3.2}$$

As a feed-forward neural network $\mathcal{N}$ may have length l of layers, the computation by $\mathcal{N}$ is essentially a composition of different activation functions associated to their corresponding layers:

$$f(\boldsymbol{x}) = (f^{(l)} \circ f^{(l-1)} \circ \cdots \circ f^{(1)})(\boldsymbol{x}). \tag{3.3}$$

In this case, $f^{(1)}$ is called the *input layer* of the network $\mathcal{N}$, $f^{(l)}$ is called the *output layer*, and all the layers in between are called the *hidden layers*.

Activation Functions The concept of hidden layers requires the choice of activation functions, for which we list some common ones below.

- Rectified linear unit (ReLU) [19, 29, 48]: $\mathrm{ReLU}(x) = \max\{x, 0\}$;
- Logistic sigmoid: $\sigma(x) = \dfrac{1}{1 + e^{-x}}$;
- Hyperbolic tangent: $\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$;

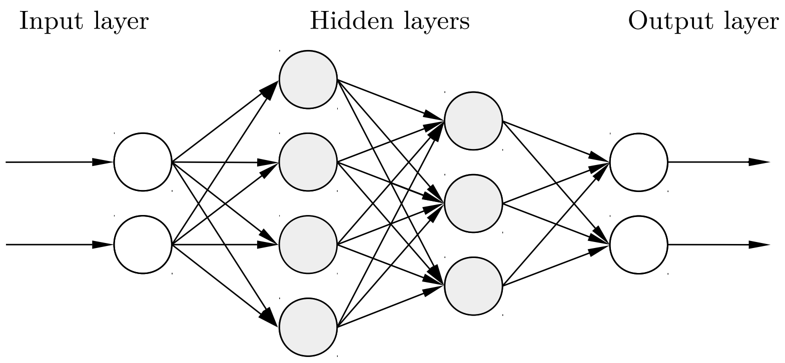


Figure 3.1: A fully-connected *feed-forward* neural network. It comprises four layers: an input layer, two hidden layers (grey), and an output layer. The information flows through the layers from left to right without any feedback connections.

- Softmax: $\mathrm{softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$ for $i = 1, \ldots, n$.

Note that unlike $\mathrm{ReLU}(x)$, $\sigma(x)$, and $\tanh(x)$, which are functions of one variable x from the previous layers, $\mathrm{softmax}(\boldsymbol{x})_i$ represents a probability distribution over the vector $\boldsymbol{x}$ with n possible values. It is often used in the output layer of a neural network to represent the predictive probability distribution over n possible classes, and its inputs, i.e., the raw (un-normalised) predictions from previous layers, are called the *logits*.

Example 1 (A Feed-Forward Neural Network)**.** *The* feed-forward neural network *in Figure 3.1 consists of four layers: an input layer with* 2 *neurons, two hidden layers with* 4 *and* 3 *layers respectively, and an output layer with* 2 *neurons. The layers are* fully-connected *as each neuron of the current layer receives input from every neuron of the previous layer.*

3.1.2 Convolutional Neural Networks

Convolutional neural networks (CNNs), also known as *convolutional networks*, are a specialised kind of network designed for processing data that has a grid-like topology, for instance, image data, which can be regarded as a two-dimensional grid of pixels. Here, “convolutional” indicates that a convolutional network employs a mathematical operation called *convolution*, which is a specialised kind of linear operation on two functions. In other words, CNNs are essentially neural networks that utilise convolution instead of general matrix multiplication in no less than one of the layers. The definition of convolution is as follows.

Definition 3.2 (Convolution)**.** *The* convolution *operation of two functions $f(t)$ and $g(t)$, denoted as $f * g$, is defined as*

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\mathrm{d}\tau \tag{3.4}$$

in the continuous case, and in the discrete case the integral is replaced by a sum, i.e.,

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau), \tag{3.5}$$

where, in convolutional network terminology, f is often referred to as the input*, g as the* kernel*, and $f * g$ as the* feature map*.*

Naturally, convolutions can be extended to multi-dimensional cases such as images. For example, given a 2D image $\boldsymbol{\alpha}$ as the input and a 2D kernel kernel, the feature map extracted from the 2D convolution is

$$(\boldsymbol{\alpha} * \mathsf{kernel})(i,j) = \sum_m \sum_n \boldsymbol{\alpha}(m,n) \cdot \mathsf{kernel}(i-m, j-n), \tag{3.6}$$

and since convolution is commutative, the feature map can be equivalently written as

$$(\mathsf{kernel} * \boldsymbol{\alpha})(i,j) = \sum_m \sum_n \boldsymbol{\alpha}(i-m, j-n) \cdot \mathsf{kernel}(m,n). \tag{3.7}$$

However, in general neural network libraries do not use convolution, but instead implement a related function named the *cross-correlation*

$$(\boldsymbol{\alpha} \star \mathsf{kernel})(i,j) = \sum_m \sum_n \boldsymbol{\alpha}(i+m, j+n) \cdot \mathsf{kernel}(m,n). \tag{3.8}$$

The key difference between these two equations, i.e., $-$ or $+$, determines whether the kernel is flipped and what pixels of the image are processed for each element of the feature map. Specifically, the kernel is flipped in convolution but not in cross-correlation; and the image is traversed bottom-up/right-left in convolution whilst top-down/left-right in cross-correlation. In a machine learning context, often cross-correlation is implemented but called convolution, which does not affect the

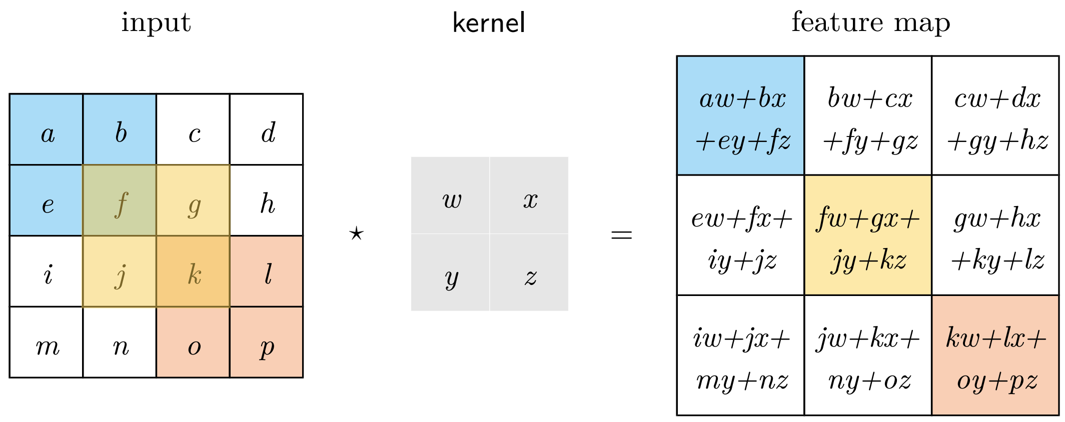


Figure 3.2: Illustration of a 2D *convolution* operation with kernel size 2 × 2. Here the convolution is "valid" as the feature map is restricted to positions where the kernel lies only within the input matrix. Same colour indicates how the element of the feature map is formed by applying the kernel to the corresponding region of the input.

network's performance as the weights are learned flipped, thus we just follow the convention of referring to both operations as convolution.

Example 2 (Convolution)**.** *The 2D* convolution *operation shown in Figure 3.2 takes a* 4×4 *input matrix* $\boldsymbol{\alpha}$ *and a* 2×2 kernel*. From Equation (3.8) we have the value of the feature map in the blue region from* $(\boldsymbol{\alpha} \star \mathsf{kernel})(0,0) = \sum_{m=0}^{1}\sum_{n=0}^{1} \boldsymbol{\alpha}(m,n) \cdot \mathsf{kernel}(m,n) = \boldsymbol{\alpha}(0,0) \cdot \mathsf{kernel}(0,0) + \boldsymbol{\alpha}(0,1) \cdot \mathsf{kernel}(0,1) + \boldsymbol{\alpha}(1,0) \cdot \mathsf{kernel}(1,0) + \boldsymbol{\alpha}(1,1) \cdot \mathsf{kernel}(1,1) = aw + bx + ey + fz.$

Pooling A limitation of the feature map generated from the convolution operation is that it records the precise position of the features in the input, which means that even a very slight movement of the feature position, such as shifting, rotation, re-cropping, and other minor changes to the input image, will produce a different feature map. In some circumstance, however, it is whether the feature is present that matters instead of where exactly it is.

A common and robust approach to addressing this problem is to use a *pooling* function, which replaces the output at a certain location with a summary statistic of the nearby outputs, and therefore makes the feature map approximately invariant to

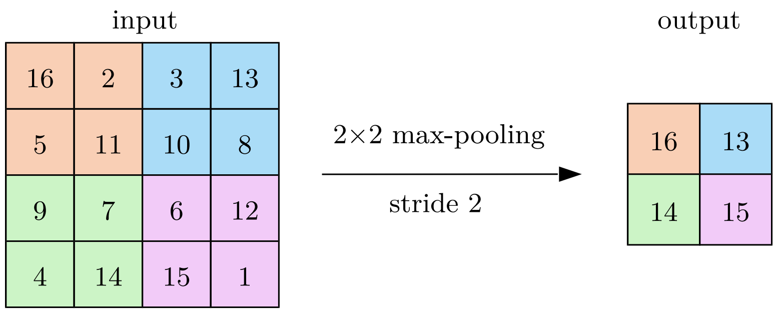


Figure 3.3: An example of the *max-pooling* operation with a 2×2 *filter* and a *stride* of 2, shown above and below the black arrow. Left: pictorial representation of a 4×4 matrix as the initial input. Right: a 2×2 output matrix after the convolution operation. Same colour indicates the output value is retrieved from the corresponding region of the input.

small translations of the input. Two popular pooling functions are the *max-pooling* and the *average-pooling* operations.

Example 3 (Max-Pooling)**.** *The* max-pooling *operation in Figure 3.3 takes a* 4×4 *matrix as an input and has a* 2×2 filter *covering each of the regions in different colours, as shown in the pictorial representation. For each region, the max-pooling operation takes the* maximum *of that region, e.g.,* $16 = \max\{16, 2, 5, 11\}$ *and* $15 = \max\{6, 12, 15, 1\}$*. The* stride *of* 2 *means that the input matrix is traversed by* $(2, 2)$ *in row and column, which in this case does not overlap regions.*

We also mention that in convolutional networks a typical layer consists of three components: convolution operations to generate a set of linear activations; a nonlinear activation function such as ReLU; and then a pooling function.

3.1.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) [61], or *recurrent networks*, are a class of neural networks where the connections between neurons form a directed graph along a temporal sequence, and thus can exhibit temporal dynamic behaviours. The capability of processing sequential data makes them applicable to tasks such as machine translation, speech/handwriting recognition, and protein homology detection.

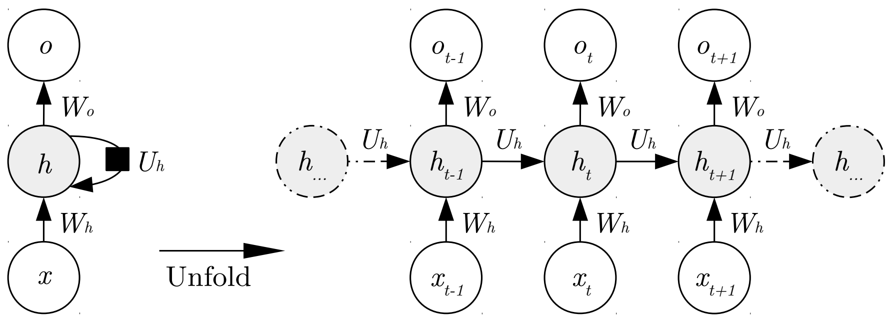


Figure 3.4: The time-unfolded computational graph of a recurrent network. Left: the RNN with input-to-hidden weight matrix $\boldsymbol{W}_h$, hidden-to-hidden recurrent connections parametrised by weight matrix $\boldsymbol{U}_h$, and hidden-to-output weight matrix $\boldsymbol{W}_o$. The black square indicates a delay of a single time instance. Right: the same RNN as an unfolded computational graph in which each neuron is associated with a time instance t.

Below we introduce the definition of recurrent neural networks. Note that while there are different design patterns, we choose a reasonably representative recurrent network that has recurrent connections between hidden units and produces an output at each time instance, illustrated in Figure 3.4. Other possible variations include recurrent networks that have recurrent connections only from the output at one time instance to the hidden unites at the next time instance, or that read an entire input sequence then produce a single output at the very end.

Definition 3.3 (Recurrent Neural Networks)**.** *A* recurrent neural network *is a network that has the following forward propagation equations at each time instance t:*

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{U}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}_h), \tag{3.9}$$

$$\boldsymbol{o}_t = \mathrm{softmax}(\boldsymbol{W}_o \boldsymbol{h}_t + \boldsymbol{b}_o), \tag{3.10}$$

where $\boldsymbol{W}_h$ *(input-to-hidden),* $\boldsymbol{U}_h$ *(hidden-to-hidden), and* $\boldsymbol{W}_o$ *(hidden-to-output) are the weight matrices;* $\boldsymbol{b}_h$ *and* $\boldsymbol{b}_o$ *are the bias vectors; and* tanh *and* softmax *can be other activation functions.*

Example 4 (RNN Unfolding)**.** *The RNN in Figure 3.4 maps an input sequence* $\boldsymbol{x}$ *to a corresponding output sequence* $\boldsymbol{o}$ *of the same length, with recurrent* *connections between the hidden units $\boldsymbol{h}$ (grey) indicated by the self-loop. From the unfolded computational graph on the right, we observe that, at time instance t, the value of the hidden neuron $\boldsymbol{h}_t$ is affected by the current input $\boldsymbol{x}_t$ and also $\boldsymbol{h}_{t-1}$ computed at the previous time instance $t-1$.*

Whilst the recurrent networks introduced above can connect previous information to the present task, e.g., to predict the last word in a short sentence “Today the sky is quite blue”, unfortunately when the gap between the relevant information and the place where it is needed grows, e.g., to predict the last word in a relatively long paragraph “Air pollution is getting much worse in this region... Today the sky is quite grey”, the standard recurrent networks are incapable of learning such long distance connections between information. The underlying challenge of learning *long-term dependencies* is that the gradients propagated over many stages tend to either vanish or explode.

An effective approach to having derivatives that neither vanish nor explode is the idea of *gated* RNNs, with connection weights that may change at each time instance. This allows the network to *accumulate* information over a long duration, such as to hold “air pollution” as evidence for the subsequent “sky” prediction, and once the information has been used, it might be beneficial for the network to *forget* the old state. Instead of manually choosing when to accumulate or to forget, gated RNNs can learn to decide when to do it. Below we introduce one of the most successful gated RNNs, that is, long short-term memory.

Long Short-Term Memory As a recurrent network architecture, *long short-term memory* [25], also known as LSTM, is capable of learning long-term dependencies. The core contribution of LSTM is the introduction of a self-loop with gated weight, by which the time scale of integration can change based on the input sequence. The LSTM models have been proved particularly successful in various practical applications, such as handwriting generation, image captioning, and parsing.

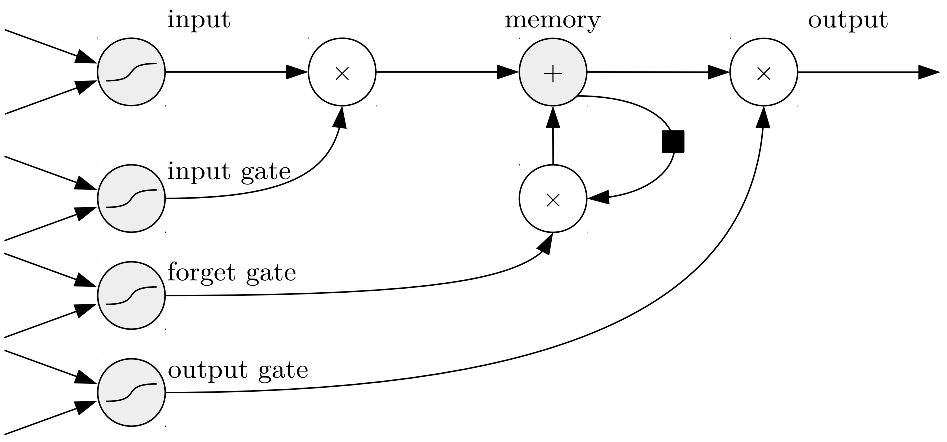


Figure 3.5: A long short-term memory unit that consists of an *input gate*, a *forget gate*, an *output gate*, and a *memory* cell. Whereas the input unit may have any squashing non-linearity, e.g., σ (drawn) or tanh, the gating units all have a sigmoid non-linearity. The memory cell has a linear self-loop, indicated by the black square as a delay of a single time step, whose weight is controlled by the forget gate.

Definition 3.4 (Long Short-Term Memory)**.** *A common* long short-term memory *unit comprises an* input gate*, a* forget gate*, an* output gate*, and a* memory *cell, as shown in Figure 3.5. Its forward propagation equations are as follows:*

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i\boldsymbol{x}_t + \boldsymbol{U}_i\boldsymbol{h}_{t-1} + \boldsymbol{b}_i), \tag{3.11}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f\boldsymbol{x}_t + \boldsymbol{U}_f\boldsymbol{h}_{t-1} + \boldsymbol{b}_f), \tag{3.12}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o\boldsymbol{x}_t + \boldsymbol{U}_o\boldsymbol{h}_{t-1} + \boldsymbol{b}_o), \tag{3.13}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \sigma(\boldsymbol{W}_c\boldsymbol{x}_t + \boldsymbol{U}_c\boldsymbol{h}_{t-1} + \boldsymbol{b}_c), \tag{3.14}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \tag{3.15}$$

where $\boldsymbol{W}_i$, $\boldsymbol{U}_i$, and $\boldsymbol{b}_i$ denote respectively the input weights, recurrent weights, and biases for the external input gate $\boldsymbol{i}_t$*; $\boldsymbol{W}_f$, $\boldsymbol{U}_f$, and $\boldsymbol{b}_f$ denote those for the* forget gate $\boldsymbol{f}_t$*; $\boldsymbol{W}_o$, $\boldsymbol{U}_o$, and $\boldsymbol{b}_o$ denote those for the* output gate $\boldsymbol{o}_t$*; and $\boldsymbol{W}_c$, $\boldsymbol{U}_c$, and $\boldsymbol{b}_o$ denote those for the* memory *cell $\boldsymbol{c}_t$.*

The block diagram of the LSTM unit is illustrated in Figure 3.5. Starting from the top left, the “input” feature can be computed by a regular artificial neuron unit with any squashing non-linearity such as σ (drawn in the figure) or tanh. All the three gating units, namely the "input gate", the "forget gate", and the "output gate", have their own input weights $\boldsymbol{W}$, recurrent weights $\boldsymbol{U}$, and biases $\boldsymbol{b}$, as well as a sigmoid non-linearity σ. The input value can be accumulated into the "memory" cell if the input gate allows it. As the most important component of the LSTM, the memory cell has a linear self-loop with weight controlled by the forget gate that sets a value from $(0,1)$ via σ to this weight. Here, the black square on the self-loop indicates a delay of a single time instance. The output of the LSTM unit can be shut off by the "output gate". Apart from this, among the variants of the LSTM unit, the memory cell can be utilised as an extra input, with three additional weights, into the three gating units (not drawn in the figure to avoid confusion).

3.1.4 Classification

Deep neural networks can be deployed to solve various tasks, e.g., the *classification* problem. In this type of task, a network $\mathcal{N}$, whether it is feed-forward, convolutional, or recurrent, is asked to specify which of n classes an input belongs to. Specifically, we view $\mathcal{N}$ as having a computation function $f : \mathbb{R}^m \mapsto \{1, \ldots, n\}$, which maps an input $\boldsymbol{x} \in \mathbb{R}^m$ to a certain category identified by numeric code $f(\boldsymbol{x}) \in \{1, \ldots, n\}$. Alternatively, $\mathcal{N}$ can be the function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, i.e., mapping to a score (logit) probability distribution over n classes, for which the result category is $\arg\max_i f_i(\boldsymbol{x})$, where i denotes the i-th element of $f(\boldsymbol{x})$.

In this thesis, we focus on a special kind of classification task – *object recognition*, where the input $\boldsymbol{x}$ is an image or a video in the domain $\mathrm{D} = [0,1]^m$ normalised from $[0,255]^m$, and the output is a numeric code c identifying the object in the image or the video from a set of classes C. For a network $\mathcal{N}$, we use $\mathcal{N}(\boldsymbol{x}, c)$ to denote the confidence, expressed as a logit value before the softmax layer or a $\mathsf{probability}$ value after normalising the score, of $\mathcal{N}$ believing that $\boldsymbol{x}$ is in class $c \in C$. Moreover, we write $\mathcal{N}(\boldsymbol{x}) = \arg\max_{c \in C} \mathcal{N}(\boldsymbol{x}, c)$ for the category into which $\mathcal{N}$ classifies $\boldsymbol{x}$. Besides, we use $\boldsymbol{\alpha}$ to indicate an image input in Chapters 4 and 5, and $\boldsymbol{v}$ to indicate a video input in Chapter 6.

3.2 Distance Metrics and Lipschitz Continuity

3.2.1 Distance Metrics

In deep learning, sometimes we need to measure the difference between two inputs. In computer vision, such a measurement should reflect the perceptual similarity between the two inputs comparable to, for instance, human perception for image or video classification neural networks. In practice, it is more common to employ a *metric* or *distance function*.

Formally, a metric on a set D is a distance function $\mathfrak{d} : \mathrm{D} \times \mathrm{D} \mapsto [0, \infty)$ that maps the distance between two inputs $\boldsymbol{x}, \boldsymbol{x}' \in \mathrm{D}$ to a non-negative real number, and satisfies the following standard axioms of a metric space:

- $\mathfrak{d}(\boldsymbol{x}, \boldsymbol{x}') \geq 0$ (non-negativity),
- $\mathfrak{d}(\boldsymbol{x}, \boldsymbol{x}') = 0 \iff \boldsymbol{x} = \boldsymbol{x}'$ (identity of indiscernibles),
- $\mathfrak{d}(\boldsymbol{x}, \boldsymbol{x}') = \mathfrak{d}(\boldsymbol{x}', \boldsymbol{x})$ (symmetry),
- $\mathfrak{d}(\boldsymbol{x}, \boldsymbol{x}'') \leq \mathfrak{d}(\boldsymbol{x}, \boldsymbol{x}') + \mathfrak{d}(\boldsymbol{x}', \boldsymbol{x}'')$ (triangle inequality).

Since we are working essentially in a vector space, as the inputs are either images or videos, we can utilise the concept of the L^p *norm*, $p \geq 1$ and take the metric induced by a norm as $\mathfrak{d}(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_p$.

Here we briefly introduce the metrics used in the thesis, i.e., the Euclidean, the Manhattan, the Chebyshev, and the Hamming distances. The *Euclidean metric*, also known as the L^2 norm, computes the Euclidean distance between the two inputs by

$$\|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sqrt{\sum_i (\boldsymbol{x}_i - \boldsymbol{x}'_i)^2}. \tag{3.16}$$

Because the Euclidean distance is so frequently used in machine learning, it is sometimes simply denoted as $\|\boldsymbol{x} - \boldsymbol{x}'\|$. However, in this thesis different metrics are used and compared, therefore we keep the subscript 2 to avoid confusion. Speaking of other metrics, in certain applications it is important to discriminate between the elements that are exactly zero and the elements that are very small but nonzero.

In the cases when the difference between zero and nonzero elements matters, the *Manhattan distance*, i.e., the L^1 norm,

$$\|\boldsymbol{x} - \boldsymbol{x}'\|_1 = \sum_i |\boldsymbol{x}_i - \boldsymbol{x}'_i| \tag{3.17}$$

comes into place as it grows at the same rate in all dimensions but simultaneously retains mathematical simplicity. Another metric that commonly arises is the *Chebyshev distance*, i.e., the L^∞ norm, which computes the greatest difference of the two inputs along any coordinate dimension,

$$\|\boldsymbol{x} - \boldsymbol{x}'\|_\infty = \max_i |\boldsymbol{x}_i - \boldsymbol{x}'_i|. \tag{3.18}$$

Apart from the above-mentioned L^1, L^2, and L^∞ norms, sometimes we need to measure the distance between two inputs by counting the number of different vector components, that is, the number of nonzero elements in $\boldsymbol{x} - \boldsymbol{x}'$. In this case, the *Hamming distance* is used with the definition

$$\mathfrak{d}_{\mathsf{Hamming}}(\boldsymbol{x}, \boldsymbol{x}') = |\{\boldsymbol{x}_i \mid \boldsymbol{x}_i \neq \boldsymbol{x}'_i\}|. \tag{3.19}$$

Remark 1 (Hamming Distance)**.** *Although in some contexts referred to as the "L^0* norm*", the* Hamming distance *is **not** actually a norm because it is not positive homogeneous, i.e., it does not satisfy* positive homogeneity*, i.e., $\forall m \in \mathbb{R}, \|m \cdot (\boldsymbol{x} - \boldsymbol{x}')\|_p = |m| \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|_p$, like L^1, L^2, and L^∞ norms do. In other words, for the Hamming distance, scaling $\boldsymbol{x} - \boldsymbol{x}'$ by m does not change the number of nonzero elements.*

3.2.2 Lipschitz Continuity

In this thesis, we restrict neural networks to those that satisfy the *Lipschitz continuity* [52] assumption. Below is its definition in mathematical analysis.

Definition 3.5 (Lipschitz Continuity)**.** *A function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$ is* Lipschitz continuous *with respect to distance metric L^p for $p \geq 1$ if there exists a constant $\mathsf{Lip} > 0$ such that, for inputs $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^m$, we have*

$$\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|_p \leq \mathsf{Lip} \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|_p, \tag{3.20}$$

where Lip *is the* Lipschitz constant.

This property is particularly useful because it enables the quantification of the assumption that a small change in the input, i.e, $\|\boldsymbol{x} - \boldsymbol{x}'\|_p$ measured by $L^p, p \geq 1$, resulting from, for example, adversarial perturbations will have a corresponding small change in the output, i.e., $\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|_p$. More importantly, this small change is guarded and guaranteed to not exceed the upper bound $\mathsf{Lip} \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|_p$.

Note that all networks whose inputs are bounded, including all the state-of-the-art image and video classification networks we studied, are Lipschitz continuous. Specifically, it is shown in [59, 68] that most known types of layers, including fully-connected, convolutional, ReLU, max-pooling, sigmoid, softmax, etc., have Lipschitz continuous activation functions.

3.3 Safety of Deep Neural Networks

Despite the success of deep learning in many areas, serious concerns have been raised regarding the deployment of deep neural networks (DNNs) in real-world safety-critical scenarios, e.g., autonomous driving and automatic medical diagnosis. In this section, we discuss the key problem of DNN safety and present a number of properties to analyse safety. Let function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, then for $\boldsymbol{x} \in \mathbb{R}^m$, we let $f_i(\boldsymbol{x})$ denote the i-th element of the vector $f(\boldsymbol{x})$.

Definition 3.6 (Erroneous Behaviour of DNNs)**.** *Given a (trained) DNN* $\mathcal{N}$ *with the computation function* $f : \mathbb{R}^m \mapsto \mathbb{R}^n$*, a human decision oracle* $\mathcal{H} : \mathbb{R}^m \to \mathbb{R}^n$*, and a legitimate input* $\boldsymbol{x} \in \mathbb{R}^m$*, an* erroneous behaviour *of* $\mathcal{N}$ *is such that*

$$\arg\max_i f_i(\boldsymbol{x}) \neq \arg\max_i \mathcal{H}_i(\boldsymbol{x}). \tag{3.21}$$

Intuitively, an erroneous behaviour is witnessed by the existence of an input $\boldsymbol{x}$ on which the DNN and a human user have different perception.

3.3.1 Adversarial Examples

Obviously, erroneous behaviours include those training and test samples that are classified incorrectly by the model, but *adversarial examples* [68] represent another category of erroneous behaviours that have safety implications.

Definition 3.7 (Adversarial Examples). *Given a (trained) DNN $\mathcal{N}$ with the computation function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, a human decision oracle $\mathcal{H} : \mathbb{R}^m \to \mathbb{R}^n$, and a legitimate input $\boldsymbol{x} \in \mathbb{R}^m$ with $\arg\max_i f_i(\boldsymbol{x}) = \arg\max_i \mathcal{H}_i(\boldsymbol{x})$, an* adversarial example *$\boldsymbol{x}'$ of $\mathcal{N}$ is defined as:*

$$\begin{aligned} \exists\ \boldsymbol{x}' :\quad & \arg\max_i \mathcal{H}_i(\boldsymbol{x}') = \arg\max_i \mathcal{H}_i(\boldsymbol{x}) \\ & \wedge\ \arg\max_i f_i(\boldsymbol{x}') \neq \arg\max_i f_i(\boldsymbol{x}) \\ & \wedge\ \boldsymbol{x} \approx \boldsymbol{x}', \end{aligned} \tag{3.22}$$

where $\approx$ denotes semantic closeness *of two inputs.*

Intuitively, $\boldsymbol{x}$ is an input on which the DNN and a human user have the same classification and, based on this, an adversarial example is another input $\boldsymbol{x}'$ that is classified differently than $\boldsymbol{x}$ by network $\mathcal{N}$, i.e., $\arg\max_i f_i(\boldsymbol{x}') \neq \arg\max_i f_i(\boldsymbol{x})$, even when they are semantically similar, i.e., $\boldsymbol{x}' \approx \boldsymbol{x}$, and the human user believes that they should be the same, i.e., $\arg\max_i \mathcal{H}_i(\boldsymbol{x}') = \arg\max_i \mathcal{H}_i(\boldsymbol{x})$. In reality, the semantic closeness, i.e., $\approx$, between inputs is often approximated via some similarity measures such as the distance metrics (Section 3.2) or the structural similarity (SSIM) index [71].

3.3.2 Safety Properties of Neural Networks

In the following, we introduce four DNN safety properties that have been studied in the literature, namely the *local robustness property*, the *output reachability property*, the *interval property*, and the *Lipschitzian property*.

Local Robustness Property

Local robustness requires that the decision of a DNN is invariant against small perturbations. The following definition is adapted from that of [28].

Definition 3.8 (Local Robustness)**.** *Given a DNN $\mathcal{N}$ with the computation function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, and an input region $\eta \subseteq \mathbb{R}^m$, the* (un-targeted) local robustness *of $\mathcal{N}$ on η is defined as*

$$\textsc{Robust}(f, \eta) \overset{def}{=} \forall \boldsymbol{x}, \boldsymbol{x}' \in \eta : \arg\max_i f_i(\boldsymbol{x}) = \arg\max_i f_i(\boldsymbol{x}'). \tag{3.23}$$

For the targeted local robustness *of a class c, it is defined as*

$$\textsc{Robust}(f, \eta) \overset{def}{=} \forall \boldsymbol{x} \in \eta : \arg\max_i f_i(\boldsymbol{x}) \neq c. \tag{3.24}$$

Intuitively, local robustness ensures that all inputs in the region η have the same classification, and when targeted for a certain class c, it assures that none of the inputs in η is classified as c. In most circumstances, the region η is defined with respect to an input and a distance metric (Section 3.2). Below we define the verification of the local robustness property.

Definition 3.9 (Verification of Local Robustness)**.** *Given a DNN $\mathcal{N}$ with the computation function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, and an input region $\eta \subseteq \mathbb{R}^m$, the* verification of local robustness *is to determine if*

$$\textsc{Robust}(f, \eta) = \texttt{True}. \tag{3.25}$$

Output Reachability Property

Output reachability is to compute the set of outputs with respect to a given set of inputs. Formally, we follow the terminology from [59, 81] and have the following definition.

Definition 3.10 (Output Reachability)**.** *Given a DNN $\mathcal{N}$ with the computation function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, and an input region $\eta \subseteq \mathbb{R}^m$, the* output reachable set *of f and η is a set $\textsc{Reach}(f, \eta)$ such that*

$$\textsc{Reach}(f, \eta) \overset{def}{=} \{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \eta\}. \tag{3.26}$$

The output reachability problem is highly *non-trivial* due to the fact that the computation function f is highly non-linear, and also the region η is continuous, which implies the existence of an infinite number of inputs in it. Based on this, we define the following verification problem.

Definition 3.11 (Verification of Output Reachability)**.** *Given a DNN $\mathcal{N}$ with the computation function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, an input region $\eta \subseteq \mathbb{R}^m$, and an output region $\mathcal{Y} \in \mathbb{R}^n$, the* verification of output reachability *on f, η, and $\mathcal{Y}$ is to determine if*

$$\textsc{Reach}(f, \eta) = \mathcal{Y}. \tag{3.27}$$

Intuitively, the verification of output reachability is to check whether all inputs in η are mapped onto a given set $\mathcal{Y}$ of outputs, and vice versa, whether all outputs in $\mathcal{Y}$ have a corresponding $\boldsymbol{x}$ in η. As a simpler question, we might be interested in computing for a specific dimension of the set $\textsc{Reach}(f, \eta)$ its greatest or smallest value, for example, the greatest classification confidence of a specific class. We also call such a problem the *reachability problem.*

Interval Property

An *interval property* is to compute a convex over-approximation of the output reachable set. We follow the terminology from interval-based approaches, which are a typical category of methods to compute this property. Formally, we have the following definition.

Definition 3.12 (Interval Property)**.** *Given a DNN $\mathcal{N}$ with the computation function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, and an input region $\eta \subseteq \mathbb{R}^m$, the* interval property *of f and η is to compute a convex set $\textsc{Interval}(f, \eta)$ such that*

$$\textsc{Interval}(f, \eta) \supseteq \{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \eta\}. \tag{3.28}$$

And, $\textsc{Interval}(f, \eta)$ contains the convex hull*, i.e., the* smallest *convex set containing the set $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \eta\}$.*

While the computation of such a set can sometimes be trivial, e.g., in classification tasks when the output $f(\boldsymbol{x})$ is often a positive integer representing a certain class, and the total amount of classes is finite, it is expected that $\textsc{Interval}(f, \eta)$ is as close as possible to $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \eta\}$, i.e., ideally it is a convex hull. Intuitively, $\textsc{Interval}(f, \eta)$ is an over-approximation of $\textsc{Reach}(f, \eta)$. Based on this, we can define the following verification problem.

Definition 3.13 (Verification of Interval Property)**.** *Given a DNN $\mathcal{N}$ with the computation function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, an input region $\eta \subseteq \mathbb{R}^m$, and an output region $\mathcal{Y} \in \Re^n$ given as a convex set, the* verification of an interval property *on f, η, and $\mathcal{Y}$ is to determine if*

$$\mathcal{Y} \supseteq \textsc{Reach}(f, \eta). \tag{3.29}$$

Intuitively, the verification of an interval property is to check whether all inputs in η are mapped into $\mathcal{Y}$. Similarly to the reachability property, we might also be interested in simpler problems, e.g., determining whether a given real number d is a valid upper bound for a specific dimension of $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \eta\}$.

Lipschitzian Property

A *Lipschitzian property*, inspired by the Lipschitz continuity (Section 3.2.2), is to monitor the changes of the outputs with respect to small changes of the inputs.

Definition 3.14 (Lipschitzian Property)**.** *Given a DNN $\mathcal{N}$ with the computation function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, an input region $\eta \subseteq \mathbb{R}^m$, and the L^p norm for $p \geq 1$,*

$$\textsc{Lipschitz}(f, \eta, L^p) \equiv \sup_{\boldsymbol{x}, \boldsymbol{x}' \in \eta} \frac{\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|_p}{\|\boldsymbol{x} - \boldsymbol{x}'\|_p} \tag{3.30}$$

is a Lipschitzian metric *of f, η, and L^p.*

Intuitively, the value of this metric is the best Lipschitz constant. Regarding this, we have the following verification problem.

Definition 3.15 (Verification of Lipschitzian Property)**.** *Given a Lipschitzian metric $\textsc{Lipschitz}(f, \eta, L^p)$ and a real value $\mathsf{Lip} \in \mathbb{R}$, the* verification of a Lipschitz property *is to determine whether*

$$\textsc{Lipschitz}(f, \eta, L^p) \leq \mathsf{Lip}. \tag{3.31}$$

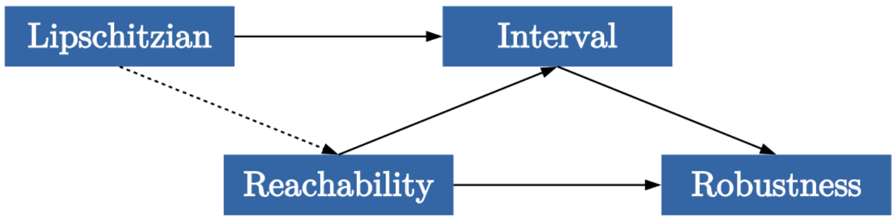


Figure 3.6: Relationship between different safety properties of neural networks. An arrow from a property A to another property B represents the existence of a simple computation to enable the computation of B based on A. The dashed arrow from "Lipschitzian" to "Reachability" means that the computation is more involved.

Relationship between Safety Properties

We illustrate the relationship between the above-mentioned four properties in Figure 3.6. To be more specific, given a Lipschitzian metric $\textsc{Lipschitz}(f, \eta, L^p)$ and a region $\eta = \eta(\boldsymbol{x}, L^p, d)$, we can compute an interval $\textsc{Interval}(f, \eta) =$

$$[f(\boldsymbol{x}) - \textsc{Lipschitz}(f, \eta, L^p) \cdot d,\ f(\boldsymbol{x}) + \textsc{Lipschitz}(f, \eta, L^p) \cdot d]\,. \tag{3.32}$$

It can be checked whether $\textsc{Interval}(f, \eta) \supseteq \{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \eta\}$ holds. Given an interval $\textsc{Interval}(f, \eta)$ or an output reachable set $\textsc{Reach}(f, \eta)$, we can check their respective robustness by determining the following expressions:

$$\textsc{Interval}(f, \eta) \subseteq \mathcal{Y}_c = \{\boldsymbol{y} \in \mathbb{R}^n \mid \arg\max_i \boldsymbol{y}_i = c\}, \text{ for some } c \tag{3.33}$$

$$\textsc{Reach}(f, \eta) \subseteq \mathcal{Y}_c = \{\boldsymbol{y} \in \mathbb{R}^n \mid \arg\max_i \boldsymbol{y}_i = c\}, \text{ for some } c \tag{3.34}$$

where $\boldsymbol{y}_i$ denotes the i-element of the output vector $\boldsymbol{y}$. The relationship between $\textsc{Reach}(f, \eta)$ and $\textsc{Interval}(f, \eta)$ is actually an implication. Moreover, we use a dashed arrow from $\textsc{Lipschitz}(f, \eta, L^p)$ to $\textsc{Reach}(f, \eta)$ to indicate that the computation is more involved by, e.g., algorithms from [59, 75, 76].

3.4 Images and Videos

3.4.1 Tensors

A tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_M}$ in an M-dimensional space is a mathematical object that has $\prod_{k=1}^{M} I_k$ components and obeys certain transformation rules. Intuitively, tensors are generalisations of vectors (i.e., one index) and matrices (i.e., two indices) to an arbitrary number of indices. Many state-of-the-art deep learning libraries, such as Tensorflow [44] and PyTorch [54], are utilising tensors to parallelise the computation with GPUs.

We introduce the following operations on tensors that are used in Chapter 4.

Definition 3.16 (Mode-m Unfolding and Folding)**.** *Given a tensor* $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_M}$*, the* mode-$m$ unfolding *of* $\mathcal{T}$ *is a matrix* $\mathbf{U}_{[m]}(\mathcal{T}) \in \mathbb{R}^{I_m \times I_N}$ *such that* $I_N = \prod_{k=1, k\neq m}^{M} I_k$*, and* $\mathbf{U}_{[m]}(\mathcal{T})$ *is defined by the mapping from element* $(i_1, \ldots, i_M)$ *to* (i_m, j)*, with*

$$j = \sum_{\substack{k=1 \\ k\neq m}}^{M} \left[i_k \times \prod_{\substack{n=k+1 \\ n\neq m}}^{M} I_n \right]. \tag{3.35}$$

Similarly, the tensor folding $\mathbf{F}$ *folds an unfolded tensor back from a matrix into a full tensor. Tensor unfolding and folding are dual operations and link tensors and matrices.*

Note that $\mathbf{U}_{[m]}(\mathcal{T})$ is a matrix because $\mathbb{R}^{I_N}$ is an one-dimensional array such that I_N is an integer. For example, to unfold a tensor $\mathcal{T} \in \mathbb{R}^{4\times3\times2}$, its mode-1 unfolding is $\mathbf{U}_{[1]}(\mathcal{T}) \in \mathbb{R}^{4\times(3\cdot2)} = \mathbb{R}^{4\times6}$, its mode-2 unfolding $\mathbf{U}_{[2]}(\mathcal{T}) \in \mathbb{R}^{3\times8}$, and its mode-3 unfolding $\mathbf{U}_{[3]}(\mathcal{T}) \in \mathbb{R}^{2\times12}$.

3.4.2 Feature Extraction

Natural data, such as natural images and sounds, forms a high-dimensional manifold, which embeds tangled manifolds to represent their inherent *features* [9]. For example, in an input $\boldsymbol{x} \in \mathbb{R}^m$, each element $x_i \in \boldsymbol{x}$ can be regarded as a feature, though not necessarily an accurate one. When performing analysis of complex data with potentially redundant information, such as the repetitiveness of images presented as pixels, such an approach generally requires expensive yet possibly unnecessary computation power and memory.

In deep learning, pattern recognition, and particularly image processing, *feature extraction* is to derive non-redundant and informative features from the initial set of measured data, thus facilitating subsequent interpretations. For instance, in the applications of image processing, various feature extraction methods are used to detect and isolate desired shapes or portions, i.e., features, of a digitised image or a video stream. Based on the underlying methodologies, we mention a few categories that are relevant to this thesis:

– “neural networks” based: A CNN can learn very rich feature representations of diverse images. To leverage the power of CNNs, a simple way is to use a pretrained CNN as a feature extractor. For instance, after passing an image into a CNN, the obtained softmax logits can be regarded as the features.

– “feature description” based: In computer vision, the *scale-invariant feature transform* (SIFT) [41] is a well-known feature extraction algorithm to describe local features in images. Irrelevant to image classifier, its extracted features have a set of invariant characteristics, such as scaling, rotation, translation, and local geometric distortion. Partially inspired by it, the standard version of *speeded up robust features* (SURF) [2, 3] is several times faster and more robust against various image transformations than SIFT, as claimed by the authors.

– “thresholding” based: Originally from signal processing and popular for cluster analysis in data mining, k-means clustering [73] partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

3.4.3 Optical Flow

In order to capture the dynamic characteristics of the moving objects in a video, we utilise *optical flow* [6, 72], which is a pattern of the apparent motion of the objects, surfaces, and edges in a sequence of frames caused by the relative movement between the scene or an observer such as a camera. It is a 2D vector field where each vector is a displacement vector showing the movement of points from the previous frame to the next. Optical flow works on two assumptions:

- the pixel intensities of an object do not change between consecutive frames,
- and neighbouring pixels have similar motion.

Definition 3.17 (Optical Flow Equation)**.** *Consider a pixel* $\mathsf{Pix}(x, y, t)$ *in a frame, where* x, y *denote the horizontal and vertical positions respectively, and* t *denotes the time dimension. If after* $\mathrm{d}t$ *time, the pixel moves by distance* $(\mathrm{d}x\,, \mathrm{d}y)$ *in the next frame, then subject to the above assumptions, the following* brightness constancy constraint

$$\mathsf{Pix}(x, y, t) = \mathsf{Pix}(x + \mathrm{d}x\,, y + \mathrm{d}y\,, t + \mathrm{d}t) \tag{3.36}$$

holds. After taking Taylor series approximation of the right-hand side, removing common terms, and dividing by $\mathrm{d}t$*, the* Optical Flow Equation *is*

$$f_x u + f_y v + f_t = 0, \tag{3.37}$$

such that

$$f_x = \frac{\partial f}{\partial x}, \quad f_y = \frac{\partial f}{\partial y}, \quad u = \frac{\partial x}{\partial t}, \quad v = \frac{\partial y}{\partial t}, \tag{3.38}$$

where f_x, f_y, f_t *are the derivatives of the image at* (x, y, t) *in the corresponding directions, i.e., horizontal, vertical, and time, and* (u, v) *are the* x *and* y *components of the* optical flow *of* $\mathsf{Pix}(x, y, t)$*.*

The optical flow equation itself has two unknowns and thus cannot be solved as such. To solve it, some additional constraints are needed, which result in another set of equations. More specifically, all optical flow methods introduce some additional conditions to estimate the actual flow. In the computer vision community, there are various differential methods to determine optical flow, such as the Lucas-Kanade method [42], the Horn-Schunck method [26], and the Gunnar Farnebäck algorithm [17].

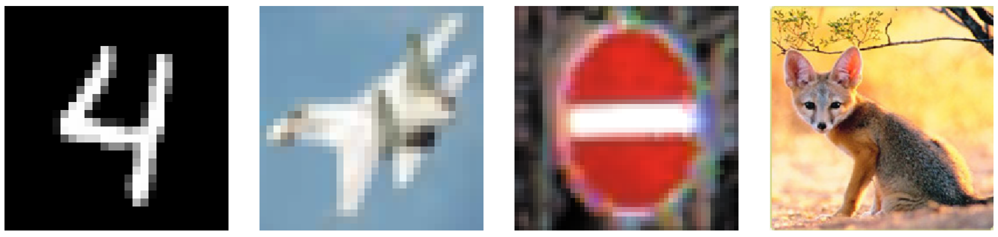


Figure 3.7: Samples from four image benchmark datasets: (a) MNIST digit "4", (b) CIFAR-10 "airplane", (c) GTSRB "no entry", and (d) ImageNet "kit fox". Though shown in the same size, these datasets actually have images of different dimensions. For example, MNIST has size $28 \times 28 \times 1$ whereas ImageNet can be $224 \times 224 \times 3$ or $299 \times 299 \times 3$ depending on the classification networks.

3.4.4 Datasets

This section introduces the standard benchmark image and video datasets.

Image Datasets

In this thesis, we perform experiments on four image datasets - MNIST, CIFAR-10, GTSRB, and ImageNet - as illustrated in Figure 3.7. They are standard benchmark datasets for adversarial attacks on DNNs, and are widely adopted by all the baseline methods compared in this thesis.

- MNIST[1] (Modified National Institute of Standards and Technology) [37] – is a grey-scale image dataset of handwritten digits from "0" to "9", which contains a training set of 60 000 examples and a test set of 10 000 examples. Each sample has dimension 28×28. The digits have been size-normalised and centred in a fixed-size image.
- CIFAR-10[2] (Canadian Institute For Advanced Research) [33] – is an image dataset of 10 mutually exclusive classes, i.e., "airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", and "truck". It consists of 60 000 32×32 colour images - 50 000 for training, and 10 000 for testing.

[1] `http://yann.lecun.com/exdb/mnist/`
[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

- GTSRB[3] (The German Traffic Sign Recognition Benchmark) [66] – is an image dataset of real-world traffic sign instances that are unique within the dataset, which means each physical traffic sign occurs once only. It contains more than 50 000 images covering 43 classes, e.g., "stop", "go right or straight", and "speed limit 80 mph".

- ImageNet[4] [13] – is a large image dataset which has 21 841 classes, falling into high-level categories such as "mammal", "furniture", and "geological formation", and consists of 14 197 122 hand-annotated and quality-controlled images. Since 2010, the ImageNet project runs the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [62], where convolutional networks such as AlexNet [34] achieved high performance in terms of correctly classifying the objects in the dataset.

Video Datasets

As a popular benchmark for human action recognition in videos, *UCF101*[5] [65], an extension of UCF50, consists of 101 annotated action classes divided into five types: 1) human-object interaction, e.g., "Soccer Juggling", "Hammering", and "Pizza Tossing"; 2) body-motion only, e.g., "Handstand Pushups", "Swing", and "Baby Crawling"; 3) human-human interaction, e.g., "Hair Cut", "Band Marching", and "Salsa Spins"; 4) playing musical instruments, e.g., "Playing Piano", "Drumming", and "Playing Flute"; and 5) sports, e.g., "Floor Gymnastics", "Biking", and "Front Crawl". It labels 13 320 video clips of 27 hours in total, and each frame has dimension $320 \times 240 \times 3$.

[3] `http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset`
[4] `http://www.image-net.org/`
[5] `https://www.crcv.ucf.edu/data/UCF101.php`

3.5 Summary

This chapter presents some technical background materials needed to understand this thesis. For instance, the convolutional neural networks are utilised to perform image classification tasks in Chapters 4 and 5, whereas Chapter 6 deploys recurrent neural networks to classify videos. Moreover, in Chapters 5 and 6, the upper and lower bounds are proved to converge by the Lipschitz continuity of the networks. Also, the unfolding and folding operations of the tensors are needed for the tensor-based parallelisation algorithm in Chapter 4.

4 Robustness of Deep Neural Networks on Pixel-Level Images

Contents

In this chapter, we consider the *local* robustness problem (Definition 3.8) of the neural networks on pixel-level images. Specifically, we define a *metric ball* to measure the neighbourhood of an input, and the *maximum safe radius* to quantify local robustness. We remark that these two concepts are used throughout this thesis. Subsequently, we extend the concept of local robustness on an input to *global* robustness on a dataset of inputs, thus enabling the evaluation of a network on an entire dataset instead of just a single image, and accordingly extend the maximum safe radius to the *expected maximum safe radius.*

Regarding the distance metrics, in this chapter we exploit the Hamming distance, introduced in Section 3.2, to measure the discrepancy between an original input and the manipulated one. The Hamming distance is an "interesting" yet "challenging" metric – "interesting" in the sense that it can intuitively and straightforwardly reflect the perturbations imposed on an image, e.g., a tiny mud speckle on a traffic sign covering ten pixels; "challenging" in the sense that it is non-differentiable, unlike the L^p norms where $p \geq 1$, and therefore most optimisation algorithms utilising gradient descent do not work in this case. In reality, different distance functions may reflect various classes of physically plausible distortions which often have their own characteristics. For example, the Hamming distance should work well on "camera occlusion" as the shape and the size of the occlusion tend to remain unchanged, whereas the Chebyshev distance (L^∞ norm) may be more appropriate to measure the discrepancy caused by "brightness change", since the brightness in all the pixels normally changes simultaneously by the same magnitude. We will cover the L^p norms in Chapters 5 and 6.

We begin the chapter in Section 4.1 by introducing the formulations of the local/global robustness. In order to compute the (expected) maximum safe radius, in Section 4.2, we define the subspace sensitivity and its tensor-based computation algorithm. Then, in Section 4.3 we utilise the computed subspace sensitivity to generate the upper and lower bounds, and provide guarantees that the bounds would converge. Finally, we implement the algorithms into the tool DeepTRE, and report the experimental results in Section 4.4.

4.1 Robustness on Pixel-Level Images

From the discussion of the DNN safety properties in Section 3.3, we know that *local robustness* is defined as the invariance of a network's classification over a small neighbourhood of a given input. In Definitions 3.8 and 3.9, the neighbourhood is simply represented by the "region η", as it can be defined with respect to an input and various similarity measures such as the structural similarity (SSIM) index.

In this thesis, we exploit the distance metrics introduced in Section 3.2 to measure the neighbourhood of an input. Formally, we define it as a *metric ball* around an input as follows.

Definition 4.1 (Metric Ball)**.** *Given an input $\boldsymbol{\alpha}$, a distance metric $\mathfrak{d}$, and a distance d, the* metric ball

$$\mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}, d) = \{\boldsymbol{\alpha}' \mid \mathfrak{d}(\boldsymbol{\alpha}, \boldsymbol{\alpha}') \leq d\} \tag{4.1}$$

is the set of inputs whose distance to $\boldsymbol{\alpha}$ is no greater than d based on the metric $\mathfrak{d}$.

Intuitively, the metric ball Ball with centre $\boldsymbol{\alpha}$ and radius d limits adversarial perturbations to at most d with respect to the distance metric $\mathfrak{d}$. When the metric is a norm, i.e., one of the L^p norms, we also refer to $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$ as a *norm ball*.

Following this, the *verification of local robustness* in Definition 3.9 can be understood as to guarantee the non-existence of adversarial examples in this norm ball, i.e., to determine whether $\textsc{Robust}(\mathcal{N}, \mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}, d)) = \texttt{True}$ holds. However, this definition returns only `True` or `False`. In this thesis, we take a step further to quantify the measurement of robustness. That is, we compute the distance to the original input in the sense that, if exceeding the distance, there definitely exists an adversarial example, whereas, within the distance, all the points are safe. We formally define this distance as the *maximum safe radius* as follows. The graphical illustration is in Figure 4.1.

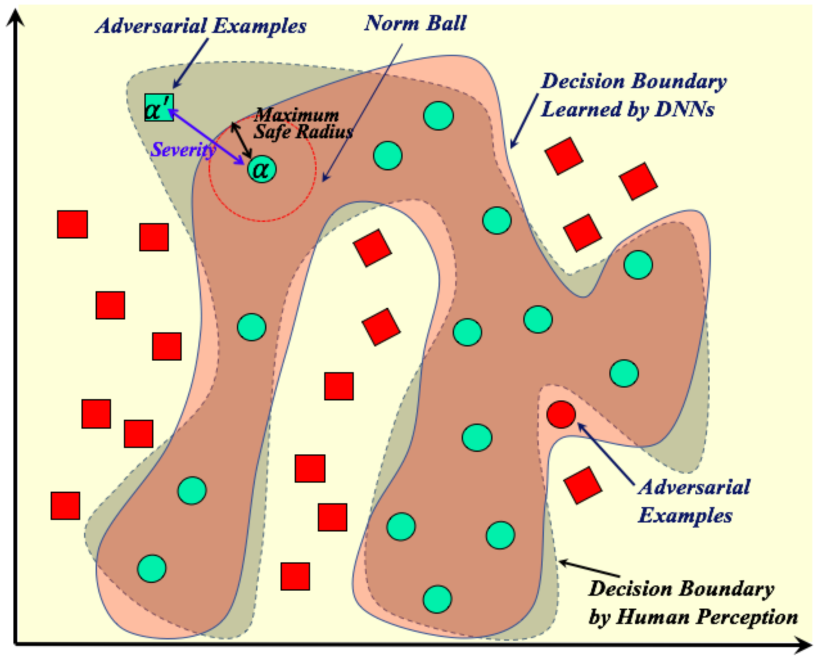


Figure 4.1: The *maximum safe radius* (MSR) problem aims to quantify the minimum distance from an original image $\boldsymbol{\alpha}$ to an adversarial example $\boldsymbol{\alpha}'$, equivalent to finding the radius of a maximum safe metric ball. The solid line represents the classification boundary learned by a neural network, while the dashed line is the decision boundary. Adversarial examples tend to lie where the decision and classification boundaries do not align. Intuitively, finding an adversarial example (green square) can only provide a loose upper bound of MSR.

Definition 4.2 (Maximum Safe Radius)**.** *Given a network $\mathcal{N}$, an input $\boldsymbol{\alpha}$, a distance metric $\mathfrak{d}$, and a distance d, the* maximum safe radius (MSR) *problem is to compute the minimum distance from the original input $\boldsymbol{\alpha}$ to an adversarial example $\boldsymbol{\alpha}'$ within the metric ball* $\mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}, d)$*, i.e.,*

$$\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}, d) = \min_{\boldsymbol{\alpha}' \in \mathrm{D}} \{\mathfrak{d}(\boldsymbol{\alpha}, \boldsymbol{\alpha}') \mid \boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}, d) \ s.t.\ \mathcal{N}(\boldsymbol{\alpha}') \neq \mathcal{N}(\boldsymbol{\alpha})\}. \tag{4.2}$$

If $\boldsymbol{\alpha}'$ does not exist in Ball*, we let* $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}, d) = d + \epsilon$*, i.e., d plus a small value ϵ.*

Intuitively, this property means that any input with distance greater than MSR, measured by metric $\mathfrak{d}$, *may* be misclassified by $\mathcal{N}$, whereas all the inputs with distance less than MSR are *guaranteed* to be classified correctly. Therefore, the key point to compute or approximate $\mathtt{MSR}$ is to find the adversarial example $\boldsymbol{\alpha}'$ with the minimal distance to $\boldsymbol{\alpha}$. Here, for simplicity, we only highlight $\mathcal{N}(\boldsymbol{\alpha}') \neq \mathcal{N}(\boldsymbol{\alpha})$ to indicate that $\boldsymbol{\alpha}'$ is an adversarial example, but remark that $\boldsymbol{\alpha}'$ meets all the constraints in the definition of adversarial examples (Definition 3.7), in the sense that the human decision oracle $\mathcal{H}$ perceives it as $\boldsymbol{\alpha}$. Besides, the semantic closeness $\boldsymbol{\alpha} \approx \boldsymbol{\alpha}'$ is reflected by the fact that $\boldsymbol{\alpha}'$ is in $\mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}, d)$. The distance threshold d is empirically dependent on the oracle. Intuitively, it should be a reasonable value because if perturbations imposed on the input are too many to the extent that even humans are not able to recognise the manipulated input, then it is no longer sensible to require a network to classify correctly. Overall, a greater $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}, d)$ indicates that network $\mathcal{N}$ is more robust in classifying input $\boldsymbol{\alpha}$ because it is less vulnerable to adversarial perturbations measured by $\mathfrak{d}$ in the neighbourhood of the input.

Regarding the distance metric $\mathfrak{d}$, we clarify that for the above definitions of local robustness $\textsc{Robust}(\mathcal{N}, \mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}, d))$ and maximum safe radius $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}, d)$, the metric can be any of the distance functions introduced in Section 3.2, i.e., the Hamming distance ($\mathfrak{d}_{\mathsf{Hamming}}$), the Manhattan distance (L^1 norm), the Euclidean distance (L^2 norm), and the Chebyshev distance (L^∞ norm). As a matter of fact, the concepts of Ball, $\textsc{Robust}$, and $\mathtt{MSR}$ are used throughout the main contributing chapters of this thesis. Particularly, in this chapter regarding the robustness on pixel-level images, we utilise the Hamming distance $\mathfrak{d}_{\mathsf{Hamming}}(\boldsymbol{\alpha}, \boldsymbol{\alpha}')$ to measure the discrepancy between the original input $\boldsymbol{\alpha}$ and its perturbed variant $\boldsymbol{\alpha}'$, while later in Chapters 5 and 6, the metric $\mathfrak{d}$ can be any of the L^p norms, denoted by $\|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_p$ with $p \geq 1$.

Global Robustness

In this chapter, apart from the verification of local robustness based on the Hamming distance, i.e., $\textsc{Robust}(\mathcal{N}, \mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{Hamming}}, d))$, quantified by the computation of $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{Hamming}}, d)$, we also generalise it to the concept of *global robustness*, which is defined as the *expected maximum safe radius* over a (finite) dataset, so that we not only know the robustness of a network with respect to a single input but also have a rough idea about the robustness of a network over a whole dataset. Note that, here the dataset indicates a set of independent and identically distributed inputs sampled from a distribution that the network is working on, e.g., the MNIST, CIFAR-10, GTSRB, and ImageNet datasets.

Formally, we define the expected maximum safe radius over a dataset as follows.

Definition 4.3 (Expected Maximum Safe Radius). *Given a network $\mathcal{N}$, a finite dataset $\mathbb{X}$, the Hamming distance metric $\mathfrak{d}_{\mathsf{Hamming}}$, and a distance d, the* expected maximum safe radius (EMSR) *problem is to compute*

$$\begin{aligned}\mathtt{EMSR}(\mathcal{N}, \mathbb{X}, \mathfrak{d}_{\mathsf{Hamming}}, d) &= \mathbb{E}_{\boldsymbol{\alpha}\in\mathbb{X}}\left[\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{Hamming}}, d)\right] && (4.3)\\ &= \frac{1}{|\mathbb{X}|}\sum_{\boldsymbol{\alpha}\in\mathbb{X}} \mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{Hamming}}, d), && (4.4)\end{aligned}$$

where $|\mathbb{X}|$ denotes the number of inputs in $\mathbb{X}$.

Intuitively, we compute the MSR for every input $\boldsymbol{\alpha}$ in the dataset $\mathbb{X}$, and take the mathematical expectation $\mathbb{E}$ with respect to the input distribution, which is approximated by averaging. See Figure 4.2 for the graphical illustration. We clarify that, while the MSR provides guarantees with respect to a single input, the EMSR does not guarantee a dataset - instead, it is used more as an empirical indication of a network's robustness on a large number of sampled inputs. For simplicity, we write $\mathfrak{d}_{\mathsf{Hamming}}$ as $\mathfrak{d}_{\mathsf{m}}$ in the remainder of this chapter.

4.2 Subspace Sensitivity

In order to compute MSR and EMSR, we define the *subspace sensitivity* of an input, which indicates how each dimension of the input as well as every possible combination of the dimensions, would affect the classification of a network. Intuitively, dimensions with higher sensitivity have a more significant influence on the network's decision-making, e.g., ideally pixels of the main object in an image, whilst dimensions with lower sensitivity tend not to count much when a network is identifying the object,

Figure 4.2: Extension from local robustness to the concept of *global robustness* is to compute the mathematical expectation of the MSR for every input in a test dataset, thus the *expected maximum safe radius*, i.e., EMSR.

e.g., the pixels in the background of an image. The analysis of subspace sensitivity enables better interpretability and explainability of neural networks in terms of how they actually "see" and/or "understand" an image.

4.2.1 Subspace for an Input

For an input $\boldsymbol{\alpha} \in \mathbb{R}^m$, we say it has m dimensions and each dimension is a real number. We are interested in the sensitivity of each dimension, as well as that of a set of certain dimensions. Thus, we define a *subspace for an input* as follows.

Definition 4.4 (Subspace for an Input)**.** *Given an input $\boldsymbol{\alpha} \in \mathbb{R}^m$ and a set of specific dimensions $T \subseteq \{1, \ldots, m\}$, the* subspace *for $\boldsymbol{\alpha}$ with respect to T is*

$$\mathsf{Sub}_{\boldsymbol{\alpha},T} = \{\boldsymbol{\alpha}' \mid \boldsymbol{\alpha}'_i \in \mathbb{R}, i \in T \text{ and } \boldsymbol{\alpha}'_j = \boldsymbol{\alpha}_j, j \in \{1, \ldots, m\} \setminus T\}, \tag{4.5}$$

where the subscript i or j denotes the i-th or the j-th dimension of $\boldsymbol{\alpha}$.

Intuitively, it means that elements in the subspace $\mathsf{Sub}_{\boldsymbol{\alpha},T}$ can take any legal value for the dimensions in the given set T, whereas they must share the same value with the original input $\boldsymbol{\alpha}$ on the other dimensions. For instance, when $T = \{m-1, m\}$, the subspace $\mathsf{Sub}_{\boldsymbol{\alpha},T}$ essentially contains all the input points with at most the last two, the $(m-1)$-th and the m-th, dimensions different from $\boldsymbol{\alpha}$, i.e., the Hamming distance $\mathfrak{d}_{\mathsf{m}}(\boldsymbol{\alpha}, \boldsymbol{\alpha}') \leq 2$.

We define the subspace for an input so that, at a later stage, we can impose adversarial perturbations on a set of dimensions in an input to assess the network's sensitivity to the overall classification. By the way, we remark that in this chapter we focus on object recognition in images, and pixel/channel values in images tend to be in range $[0, 255]$ before or $[0, 1]$ after normalisation. Therefore, in the experiments the images modified with perturbations are normally constrained to be legitimate, e.g., $\boldsymbol{\alpha}' \in [0, 255]^m$ or $\boldsymbol{\alpha}' \in [0, 1]^m$, though here we keep $\mathbb{R}^m$ for generality.

To take a step further, we are interested in not only a given set of dimensions T, but also every possible combination of the input dimensions. Regarding this, below we define the *complete set of subspaces for an input.*

Definition 4.5 (Complete Set of Subspaces for an Input)**.** *Given an input $\boldsymbol{\alpha} \in \mathbb{R}^m$ and an integer t such that $1 \leq t \leq m$, the* complete set of subspaces *for $\boldsymbol{\alpha}$ with respect to t is*

$$\mathsf{Subs}(\boldsymbol{\alpha}, t) = \{\mathsf{Sub}_{\boldsymbol{\alpha},T} \mid \forall\, T \subseteq \{1, \ldots, m\} \text{ s.t. } |T| = t\}, \tag{4.6}$$

where t indicates the number of dimensions in T.

Intuitively, it means that $\mathsf{Subs}(\boldsymbol{\alpha}, t)$ includes all the subspaces $\mathsf{Sub}_{\boldsymbol{\alpha},T}$ for any possible combination T with t dimensions. Specifically, $|\mathsf{Subs}(\boldsymbol{\alpha}, t)|$ is $\binom{m}{t} = \frac{m!}{t!(m-t)!}$. For example, when $t = 2$, then T can be any set comprising two dimensions, e.g., $\{m-1, m\}$ or $\{11, 6\}$, as long as the element in T does not exceed the total number of dimensions $\boldsymbol{\alpha}$ has, which is m. In other words, for a complete set of subspaces for an input with respect to a certain t, adversarial perturbations can be imposed upon any t number of dimensions, i.e., the Hamming distance $\mathfrak{d}_{\mathsf{m}}(\boldsymbol{\alpha}, \boldsymbol{\alpha}')$ remains at most t.

Next, we define *subspace sensitivity*, i.e., the sensitivity of a certain subspace for an input with respect to a neural network $\mathcal{N}$. Recall that, from the definition of classification in Section 3.1.4, we use $\mathcal{N}(\boldsymbol{\alpha}, c)$ to denote the confidence, either a logit value before the softmax layer or a $\mathsf{probability}$ value after normalising the score, of $\mathcal{N}$ believing that $\boldsymbol{\alpha}$ is in class c, and write $\mathcal{N}(\boldsymbol{\alpha}) = \arg\max_{c \in C} \mathcal{N}(\boldsymbol{\alpha}, c)$ for the category into which $\mathcal{N}$ classifies $\boldsymbol{\alpha}$.

Definition 4.6 (Subspace Sensitivity)**.** *Given a subspace* $\mathsf{Sub}_{\boldsymbol{\alpha},T}$ *for an input* $\boldsymbol{\alpha} \in \mathbb{R}^m$*, a neural network* $\mathcal{N}$*, the* subspace sensitivity *with respect to* $\mathsf{Sub}_{\boldsymbol{\alpha},T}$ *and* $\mathcal{N}$ *is defined as*

$$\mathsf{Sen}(\mathsf{Sub}_{\boldsymbol{\alpha},T}, \mathcal{N}) = \mathcal{N}(\boldsymbol{\alpha}, c) - \inf_{\boldsymbol{\alpha}' \in \mathsf{Sub}_{\boldsymbol{\alpha},T}} \mathcal{N}(\boldsymbol{\alpha}', c), \tag{4.7}$$

where $c = \mathcal{N}(\boldsymbol{\alpha})$ *denotes the class that* $\mathcal{N}$ *identifies* $\boldsymbol{\alpha}$*.*

Intuitively, $\mathsf{Sen}(\mathsf{Sub}_{\boldsymbol{\alpha},T}, \mathcal{N})$ is the maximal decrease of the confidence value from network $\mathcal{N}$ corresponding to the output class c that can be witnessed from the subspace $\mathsf{Sub}_{\boldsymbol{\alpha},T}$ for input $\boldsymbol{\alpha}$. In other words, given an input and a set of specific dimensions T in it, if we impose adversarial perturbations within T, we can measure the sensitivity of this T by calculating how much the network's confidence of identifying the original input $\boldsymbol{\alpha}$ as class c is reduced. If $\mathsf{Sen}(\mathsf{Sub}_{\boldsymbol{\alpha},T}, \mathcal{N})$ is larger, it means that the input dimensions in T are more sensitive regarding the decision-making of $\mathcal{N}$ on $\boldsymbol{\alpha}$ with respect to c, i.e., perturbations in T are more likely to make $\mathcal{N}$ not classify $\boldsymbol{\alpha}$ as c.

Now we extend the subspace sensitivity defined on a given set of input dimensions T to the sensitivity of a complete set of subspaces with respect to t for every input $\boldsymbol{\alpha}$ in a dataset $\mathbb{X}$. That is, given a dataset $\mathbb{X}$, a neural network $\mathcal{N}$, and an integer t such that $1 \leq t \leq m$, the sensitivity for $\mathbb{X}$ with respect to $\mathcal{N}$ and t is

$$\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t) = (\mathsf{Sen}(\mathsf{Sub}_{\boldsymbol{\alpha},T}, \mathcal{N}))_{\mathsf{Sub}_{\boldsymbol{\alpha},T} \in \mathsf{Subs}(\boldsymbol{\alpha},t), \boldsymbol{\alpha} \in \mathbb{X}}. \tag{4.8}$$

Intuitively, $\mathsf{Sub}_{\boldsymbol{\alpha},T} \in \mathsf{Subs}(\boldsymbol{\alpha}, t)$ extends from a given set of dimensions T to any set of dimensions as long as the total number of dimensions is t, and $\boldsymbol{\alpha} \in \mathbb{X}$ generalises from a single image $\boldsymbol{\alpha}$ to a dataset of images $\mathbb{X}$. Therefore, $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)$ is essentially a two-dimensional array of the maximal decreases of the confidence values from network $\mathcal{N}$ identifying class c, with one dimension corresponding to all the subspaces in $\mathsf{Subs}(\boldsymbol{\alpha}, t)$, and the other dimension corresponding to all the inputs in $\mathbb{X}$. Thus, the total number of elements in $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)$ is $\binom{m}{t} \cdot |\mathbb{X}|$.

We remark that the computation of subspace sensitivity for a dataset with respect to a certain number of dimensions is *nontrivial*, as every element in $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)$

represents an optimisation problem. For example, given a set of 20 MNIST images and letting $t = 1$, Equation (4.8) requires $\binom{28\times 28}{1} \times 20 = 15\,680$ one-dimensional optimisation problems. That is, given a dataset $\mathbb{X}$ and an integer t, the number of elements in $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)$ is in $O(|\mathbb{X}| \cdot m^t)$, i.e., polynomial in $|\mathbb{X}|$ and exponential in t. In the next section, we present a tensor-based formulation and an algorithm to solve this challenging problem via GPU parallelisation.

4.2.2 Computation of Subspace Sensitivity

In the following, we present the algorithm to compute subspace sensitivity via utilising the unfolding and folding operations of *tensors* (Definition 3.16). The basic idea of our algorithm is to transform a set of nonlinear, non-convex optimisation problems as given in Equation (4.8) into a tensor formulation, and solve this set of optimisation problems efficiently via a few parallel queries of neural networks.

Given a neural network $\mathcal{N}$, a dataset $\mathbb{X}$, and an integer t, we would like to compute the subspace sensitivity $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)$. Recall that all the inputs in the dataset are independent and identically distributed. While each input $\boldsymbol{\alpha}$ in the dataset generates a complete set of subspaces $\mathsf{Subs}(\boldsymbol{\alpha}, t)$, we remark that for different inputs $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ in $\mathbb{X}$, we have $|\mathsf{Subs}(\boldsymbol{\alpha}, t)| = |\mathsf{Subs}(\boldsymbol{\beta}, t)|$, which means that each input has the same number of subspaces for a given t. Let $p = |\mathbb{X}|$ denote the total number of inputs in the dataset, and, for each input $\boldsymbol{\alpha} \in \mathbb{R}^m$, let $k = |\mathsf{Subs}(\boldsymbol{\alpha}, t)|$ denote its total number of subspaces with respect to t, i.e, $k = \binom{m}{t}$.

Adversarial Perturbations via Grid Search

We evaluate the subspace sensitivity utilising *adversarial perturbations*. To be more specific, we impose modifications to each dimension of an input and every possible combination of the dimensions to evaluate their corresponding sensitivities in terms of how they would change the overall classification outcome of the input. Now the problem is what kind of modifications should be imposed upon an input. Ideally, every possible value in a dimension of the input should be traversed and its sensitivity for each possible value evaluated. For example, given a MNIST image with dimensions $[0, 255]^{28\times 28}$, even when $t = 1$, to evaluate sensitivity we need to try the 256 integers in all the $\binom{28\times 28}{1}$ dimensions to see which dimension with what value makes the classification confidence decrease the most. And when t increases, the computation grows exponentially. Regarding this, we define an *error tolerance* $\epsilon > 0$ as a workaround. In other words, instead of traversing every possible value of a dimension, by applying *grid search*, we recursively sample $\Delta = 256/\epsilon$ or $1/\epsilon$ (after normalising into $[0, 1]^{28\times 28}$) values in each dimension. For example, let $\epsilon = 0.1$ for a normalised MNIST image, then, for a pixel with original value 0, only $1/\epsilon = 10$ values need to be tested, i.e., $\{0.1, 0.2, \ldots, 1\}$.

Now we generalise this grid search to an input $\boldsymbol{\alpha} \in \mathbb{R}^m$ in the dataset $\mathbb{X}$. Given integer t, for a set of specific dimensions T such that $|T| = t$, there are overall Δ^t ways of imposing adversarial perturbations upon the set, i.e., Δ samples in each dimension and t dimensions in total. After adding those Δ^t perturbations, we essentially turn each subspace $\mathsf{Sub}_{\boldsymbol{\alpha},T} \in \mathsf{Subs}(\boldsymbol{\alpha}, t)$ into a two-dimensional grid $\mathbb{G}(\mathsf{Sub}_{\boldsymbol{\alpha},T}) \in \mathbb{R}^{m\times\Delta^t}$, where $\mathbb{R}^m$ indicates a manipulated input $\boldsymbol{\alpha}'$, and Δ^t the number of $\boldsymbol{\alpha}'$ on the grid. Furthermore, if we extend this grid from a subspace to the complete set of subspaces for the input $\boldsymbol{\alpha}$, and, even further, to every input in the dataset $\mathbb{X}$, then we can formulate the following tensor

$$\mathcal{T}(\mathbb{X}, t, \Delta) = \textsc{Tensor}((\mathbb{G}(\mathsf{Sub}_{\boldsymbol{\alpha},T}))_{\mathsf{Sub}_{\boldsymbol{\alpha},T}\in\mathsf{Subs}(\boldsymbol{\alpha},t),\boldsymbol{\alpha}\in\mathbb{X}}) \in \mathbb{R}^{m\times\Delta^t\times k\times p}, \tag{4.9}$$

which includes all the subspaces $\mathsf{Sub}_{\boldsymbol{\alpha},T} \in \mathsf{Subs}(\boldsymbol{\alpha}, t)$, and all the inputs $\boldsymbol{\alpha} \in \mathbb{X}$. Recall that $k = \binom{m}{t}$ and $p = |\mathbb{X}|$. Intuitively, $\mathcal{T}(\mathbb{X}, t, \Delta)$ includes all the possible adversarial perturbations, sampled by Δ, imposed on any t dimensions within every input of the dataset.

Parallelisation via Unfolding and Folding of Tensors

Subsequently, by referring to the unfolding and folding operations of tensors in Definition 3.16, we apply the mode-1 tensor unfolding to $\mathcal{T}(\mathbb{X}, t, \Delta)$ so that we get a two-dimensional matrix $\mathbf{U}_{[1]}(\mathcal{T}(\mathbb{X}, t, \Delta)) \in \mathbb{R}^{m\times N}$ such that $N = \Delta^t \cdot k \cdot p$. Note that here N is an integer indicating the total number of manipulated inputs $\boldsymbol{\alpha}' \in \mathbb{R}^m$. Then we feed this tensor into the network $\mathcal{N}$ to obtain the corresponding confidence values of $\mathcal{N}$ identifying all these perturbed $\boldsymbol{\alpha}'$ into class $c = \mathcal{N}(\boldsymbol{\alpha})$. That is,

$$Cons(\mathbb{X}, t, \Delta, \mathcal{N}) = \mathcal{N}(\mathbf{U}_{[1]}(\mathcal{T}(\mathbb{X}, t, \Delta)), c) \in \mathbb{R}^N, \tag{4.10}$$

where $N = \Delta^t \cdot k \cdot p$. Here $Cons(\mathbb{X}, t, \Delta, \mathcal{N})$ is a vector comprising N confidence values of class c, either probabilities or logits, from $\mathcal{N}$ with respect to the N perturbed inputs $\boldsymbol{\alpha}'$. We remark that Equation (4.10) only needs a single network query, and if implemented in GPUs then the method is very efficient, as demonstrated in the experimental results (Figure 4.7(c)). Now we apply a tensor folding operation to fold this vector of confidence values back to a tensor

$$\mathsf{Cons}(\mathbb{X}, t, \Delta, \mathcal{N}) = \mathbf{F}(Cons(\mathbb{X}, t, \Delta, \mathcal{N})) \in \mathbb{R}^{\Delta^t \times k \times p}. \tag{4.11}$$

Here, we highlight the difference between $\mathbb{R}^{\Delta^t \cdot k \cdot p}$ and $\mathbb{R}^{\Delta^t \times k \times p}$. Although they both indicate all the confidence values, we remark that the former is a one-dimensional array and the latter is a tensor. Intuitively, the main purpose of these three Equations (4.9), (4.10), and (4.11) is to take advantage of the GPU parallelisation via the unfolding and folding operations of tensors, because, for a neural network, classifying an input batch that contains a large number of inputs is much faster than sequentially identifying them.

Once we have obtained all the confidence values, in order to compute sensitivity defined by the maximal decrease of the confidence, we need to find the minimum confidence value so that we can measure the difference from the original confidence. Therefore, on the tensor $\mathsf{Cons}(\mathbb{X}, t, \Delta, \mathcal{N})$, we search for the minimum values along the first dimension to obtain[1]

$$\mathsf{Con}(\mathbb{X}, t, \Delta, \mathcal{N})_{\min} = \min(\mathsf{Cons}(\mathbb{X}, t, \Delta, \mathcal{N}), 1) \in \mathbb{R}^{k \times p}. \tag{4.12}$$

The reason that we search in the first dimension, marked by Δ^t, is that $\mathbb{R}^{\Delta^t}$ has covered all the possible adversarial perturbations $\boldsymbol{\alpha}'$ sampled by Δ imposed on

[1]Here, for convenience purposes, we exploit the Matlab notation $\min(Y, k)$, which computes the minimum values over the k-th dimension of a multi-dimensional array Y. We use similar notation in the remainder of this chapter.

a subspace $\mathsf{Sub}_{\boldsymbol{\alpha},T}$ such that $|T| = t$. Intuitively, for a specific set of dimensions that can be perturbed, the minimum confidence value corresponds to a particular way of manipulating the set of dimensions in the sense that it will result in the maximal confidence decrease. Actually, if we look back at the definition of subspace sensitivity in Equation (4.7), we notice that the returned minimum confidence value is exactly $\inf_{\boldsymbol{\alpha}' \in \mathsf{Sub}_{\boldsymbol{\alpha},T}} \mathcal{N}(\boldsymbol{\alpha}', c)$. Moreover, in Theorem 3, we demonstrate that grid search provides the guarantee of reaching the global minimum by utilising the Lipschitz continuity of neural networks.

To this end, we have almost solved all the $k \times p$, i.e., $\binom{m}{t} \cdot |\mathbb{X}|$, optimisation problems in Equation (4.8). Now that we already have the minimum confidence values, what is left to do is to construct a tensor of the original confidence that the network classifies the input into its class, i.e., $\mathcal{N}(\boldsymbol{\alpha}, c)$. Below is the tensor

$$\mathsf{Con}(\mathbb{X}, t, \mathcal{N}) = (\overbrace{\mathcal{N}(\boldsymbol{\alpha}, c), \ldots, \mathcal{N}(\boldsymbol{\alpha}, c)}^{k}) \in \mathbb{R}^{k \times p}, \tag{4.13}$$

which includes all the inputs $\boldsymbol{\alpha} \in \mathbb{X}$ in the dataset. Intuitively, $\mathsf{Con}(\mathbb{X}, t, \mathcal{N})$ is the tensor that contains the starting points of the optimisation problems and $\mathsf{Con}(\mathbb{X}, t, \Delta, \mathcal{N})_{\min}$ are the resulting optimal values. Note that these two tensors have the same dimension $\mathbb{R}^{k \times p}$.

The following theorem shows the correctness of our computation, where the subspace sensitivity for the whole dataset $\mathbb{X}$ with respect to the network $\mathcal{N}$ and the integer t, i.e., $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)$, has been defined in Equation (4.8).

Theorem 1 (Subspace Sensitivity)**.** *Given a dataset* $\mathbb{X}$*, a neural network* $\mathcal{N}$*, an integer* t*, and an* ϵ *grid search, the* subspace sensitivity $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)$ *for* $\mathbb{X}$ *with respect to* $\mathcal{N}$*,* t*, and* $\Delta = 1/\epsilon$ *can be computed by*

$$\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)_{\Delta} = \mathsf{Con}(\mathbb{X}, t, \mathcal{N}) - \mathsf{Con}(\mathbb{X}, t, \Delta, \mathcal{N})_{\min}. \tag{4.14}$$

Proof. By utilising the Lipschitz continuity of neural networks, Theorem 3 establishes that grid search provides the guarantee that $\mathsf{Con}(\mathbb{X}, t, \Delta, \mathcal{N})_{\min}$ reaches the global minimum. Then, combined with the definition of subspace sensitivity in Equations (4.8) and (4.7), the computation of $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)_\Delta$ via subtracting the global minimum confidence from the original confidence follows naturally. $\square$

4.3 Tensor-Based Algorithms for Upper and Lower Bounds

The computation of the maximum safe radius $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}, d)$ of Definition 4.2 is hard for the Hamming distance $\mathfrak{d}_\mathsf{m}$. Below we analyse the computational complexity in the size of input dimensions and error tolerance.

Theorem 2 (Complexity). *Given a network $\mathcal{N}$ and an input $\boldsymbol{\alpha}$ normalised into $[0,1]^m$, when the metric $\mathfrak{d}$ is the Hamming distance $\mathfrak{d}_\mathsf{m}$, the maximum safe radius problem is* NP-hard*, and there exists a deterministic algorithm that can compute $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_\mathsf{m}, d)$ in time* complexity $O((1+\frac{1}{\epsilon})^m)$ *for the worst-case scenario when the error tolerance for each dimension is $\epsilon > 0$.*

Proof. Here we consider the worst-case scenario and use a straightforward grid search to verify the time complexity needed. In the worst case, the maximum radius of a safe metric ball for network $\mathcal{N}$ is $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_\mathsf{m}, d) = m$. A grid search with grid size $\Delta = 1/\epsilon$ starts from $\mathfrak{d}_\mathsf{m} = 1$ to verify whether $\mathfrak{d}_\mathsf{m}$ is the radius of the maximum safe ball and would require the following running time in terms of the network queries.

$$\sum_{\mathfrak{d}_\mathsf{m}=1}^{m} \binom{n}{\mathfrak{d}_\mathsf{m}} \Delta^{\mathfrak{d}_\mathsf{m}} = (1+\frac{1}{\epsilon})^m. \tag{4.15}$$

$\square$

From the above proof, we obtain the following remark.

Remark 2. *Computing $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_\mathsf{m}, d)$ is a* more *challenging problem for the Hamming distance $\mathfrak{d}_\mathsf{m}$, since it requires a* higher *computing complexity than, e.g., the L^1 and L^2 norms. Namely, grid search only requires $(1/\epsilon)^m$ queries of the network to estimate $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, L^1, d))$ or $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, L^2, d)$ given* *the same error tolerance ϵ.*

4.3.1 Computation of Lower and Upper Bounds

It is demonstrated in Theorem 2 that the complexity of the maximum safe radius problem is NP-hard. Therefore, instead of directly computing the maximum safe radius of an input $\boldsymbol{\alpha}$, we propose to compute the lower and upper bounds of it, whose intuitive meanings are that any perturbed input $\boldsymbol{\alpha}'$ with distance to $\boldsymbol{\alpha}$ less than the *lower bound* is definitely safe, whereas there definitely exists an adversarial example $\boldsymbol{\alpha}'$ with distance to $\boldsymbol{\alpha}$ exceeding the *upper bound*. After obtaining the bounds, we gradually, but strictly, improve them so that eventually they *converge* to the optimal value, i.e., `MSR`. Below we present the tensor-based algorithms to compute the bounds, and later provide guarantees for their convergence.

To compute the lower and upper bounds, we utilise the definition of subspace sensitivity in Equation (4.8) and its computation in Theorem 1. Specifically, given a dataset $\mathbb{X}$, its subspace sensitivity can be computed, i.e., $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)_\Delta$. Apart from the maximum decreases of the classification confidence caused by the adversarial perturbations, we also care about the actual inputs that result in these decreases. Regarding this, we construct the *solutions*, denoted by $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_\Delta$, which is a tensor that includes all the modified inputs with the most effective adversarial manipulations, i.e., the inputs that are the most sensitive in terms of affecting the classification of the network. The way to construct the tensor of the solutions is straightforward – essentially we replace each confidence drop in $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)_\Delta$ by its corresponding perturbed input $\boldsymbol{\alpha}'$. This can be easily achieved through storing the indices during the computation procedure and then applying a few tensor operations to expand the dimension. Note that, whereas $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)_\Delta$ has dimensions $\mathbb{R}^{k \times p}$, the solutions $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_\Delta$ are in dimensions $\mathbb{R}^{m \times k \times p}$, where $\mathbb{R}^m$ indicates the perturbed input.

Now that we have the original dataset $\mathbb{X}$, the subspace sensitivity $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)_\Delta$, and the solutions $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_\Delta$, we can compute the bounds as presented below.

Recall that the integer t in the solutions constrains the number of dimensions in an input $\boldsymbol{\alpha} \in \mathbb{X}$ that can be modified. In other words, the Hamming distance between a perturbed input $\boldsymbol{\alpha}'$ and $\boldsymbol{\alpha}$ is at most t, i.e., $\mathfrak{d}_{\mathsf{m}}(\boldsymbol{\alpha}, \boldsymbol{\alpha}') \leq t$.

Lower Bounds

For an input $\boldsymbol{\alpha}$, starting from the number of dimensions $t = 1$, we compute the *lower* bound of `MSR` by finding the greatest decrease of the confidence value in its complete set of subspaces $\mathsf{Subs}(\boldsymbol{\alpha}, t)$. Note that the subspace sensitivity in Definition 4.6 is with respect to a single subspace. By comparing across the complete set of subspaces, we obtain the most sensitive subspace. And if its corresponding solution $\boldsymbol{\alpha}'$ is identified as the same as $\boldsymbol{\alpha}$, then we update the lower bound of `MSR` to t. This is because, when even the most sensitive subspace cannot alter the network's classification, none of the other subspaces can. In other words, the solutions in this complete set are all safe. Then, we gradually increase t to allow adversarial perturbations on more dimensions, and if the most sensitive solution remains correctly classified, then we simultaneously update the lower bound to t. Nevertheless, if it comes to the situation that, for a certain t, the most sensitive solution $\boldsymbol{\alpha}'$ is an adversarial example, then the maximum safe radius of this input has been obtained, i.e, $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d) = t$.

To efficiently compute the lower bounds of `MSR` for every input in the dataset using parallelisation, we extend the above algorithm from a single input $\boldsymbol{\alpha}$ to the whole dataset $\mathbb{X}$. Specifically, we reorder the solutions $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\Delta}$ and the sensitivity of the complete set of subspaces for every input in the dataset, i.e., $\mathsf{Sen}(\mathbb{X}, \mathcal{N}, t)_{\Delta}$, in a decreasing order with respect to the confidence drops in the complete set of subspaces of each input. That is, the most sensitive solution for each input is placed on top. Subsequently, we retrieve the first row of the second dimension in the sorted solutions, i.e., $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\Delta}[:, 1, :] \in \mathbb{R}^{m \times p}$, and check whether $\mathcal{N}(\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\Delta}[:, 1, :]) = \mathcal{N}(\mathbb{X})$. Intuitively, this is to check simultaneously whether the most sensitive solution for each input is an adversarial example. The returned result is an array of Boolean values, within which each `True` or `False` indicates the corresponding solution is safe or unsafe. If safe, then the lower bound of `MSR` for this input upgrades to t; and if not, then the `MSR` of this input is reached.

Finally, we remark that, after computing $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\Delta}$, no further network query is needed to compute the lower bounds.

Upper Bounds

The upper bounds are computed by iteratively accumulating perturbations based on the tensor of solutions $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\Delta}$ for every input in $\mathbb{X}$ until a misclassification occurs. To be more specific, for each input $\boldsymbol{\alpha}$ in the dataset, $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\Delta}$ contains $k = \binom{m}{t}$ number of solution points corresponding to the same amount of total subspaces in a complete set. Each solution reflects the optimal modification among the total number of Δ^t adversarial perturbations sampled by the grid search in a specific subspace, as in Equation (4.12). However, under most circumstances, a subspace-level perturbation, even the optimal one, does not tend to change the overall classification of the input immediately, especially when t is quite small. For example, given an image from the ImageNet dataset, when $t = 2$, based on the ϵ grid search, a total number of Δ^2 perturbations are sampled and imposed on each subspace that comprises 2 dimensions. For a single subspace, even when the optimal perturbation is chosen, the fact that it only changes 2 dimensions of the ImageNet image often would not change its overall classification.

As a workaround, we accumulate the sampled perturbations across multiple subspaces $\mathsf{Sub}_{\boldsymbol{\alpha},T}$, yet within the same complete set $\mathsf{Subs}(\boldsymbol{\alpha}, t)$, on $\boldsymbol{\alpha}$ to see if the perturbed input $\boldsymbol{\alpha}'$ can be turned into an adversarial example. Note that every adversarial example produces an upper bound of the `MSR`. In our case, we would like to generate the relatively smaller upper bounds so that they are closer to the actual `MSR`. We achieve this by utilising the sorted tensor of solutions $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\Delta}$. For $\boldsymbol{\alpha}$, we start modifying it from the perturbations within the more sensitive solution points, then, in the decreasing order, we gradually accumulate the modifications in the less sensitive solutions, until the manipulated $\boldsymbol{\alpha}'$ becomes an adversarial example, i.e., an upper bound of $\texttt{MSR}(\boldsymbol{\alpha}, \mathcal{N}, \mathfrak{d}_{\mathsf{m}}, d)$ is produced. However, doing this sequentially for all inputs in $\mathbb{X}$ would be inefficient, as it needs to query the network $\mathcal{N}$ after every subspace-level perturbation on each image.

We present an efficient tensor-based algorithm to generate the upper bounds of $\texttt{MSR}$ for every $\boldsymbol{\alpha} \in \mathbb{X}$ at the same time, taking advantage of GPU parallelisation. Intuitively, the primary gain provided by parallelisation is that during the computation of subspace sensitivity and the bounds, a large number of perturbed inputs are produced, depending on the values of the input dimensions and the error tolerance. And feeding all these perturbed inputs in parallel into the network will be much more efficient than classifying them sequentially. The key idea is to construct a new tensor $\mathsf{Acc} \in \mathbb{R}^{m \times k \times p}$ to maintain all the accumulated perturbations over each $\boldsymbol{\alpha}$, separately yet simultaneously.

- Initialisation: $\mathsf{Acc}[:, 1, :] = \mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\triangle}[:, 1, :]$.
- Iteratively construct the i-th row until $i = k$:

$$\mathsf{Acc}[:, i, :] = \mathsf{Acc}[:, i-1, :] \boxminus \{\mathsf{Solu}(\mathbb{X}, \mathcal{N} t)_{\triangle}[:, i, :] \Cap \mathbb{X}[:, :]\}, \tag{4.16}$$

where $\boxminus$ and $\Cap$ are tensor operations such that $\mathsf{Acc}_1 \boxminus \mathsf{Acc}_2$ imposes the non-zero elements in Acc_2 upon Acc_1, and $\mathsf{Acc}_1 \Cap \mathsf{Acc}_2$ retains those elements in Acc_1 that are different from Acc_2 and sets the other elements to 0. The two operands of these operations are required to have the same type. Intuitively, $\mathsf{Acc}[:, i, :]$ represents the result of accumulating the first i perturbations recorded in $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\triangle}[:, 1 : i, :]$ on $\mathbb{X}$.

Subsequently, we perform mode-1 unfolding to Acc and pass the result to the network $\mathcal{N}$, which yields the classifications $\mathcal{N}(\mathbf{U}_{[1]}(\mathsf{Acc})) \in \{1, \ldots, n\}^{k \cdot p}$, where n denotes the number of categories. After that, a tensor folding operation is applied to obtain $\mathbf{F}(\mathcal{N}(\mathbf{U}_{[1]}(\mathsf{Acc}))) \in \{1, \ldots, n\}^{k \times p}$. Finally, we can compute the minimum column index along each row such that a misclassification occurs, denoted by $\{n_1, \ldots, n_p\}$ such that $1 \leq n_i \leq k$ and $1 \leq i \leq p$. Then we let

$$\mathbb{X}' = \{\mathsf{Acc}_{:, n_i, :} \in \mathbb{R}^{m \times p} \mid n_i \in \{n_1, \ldots, n_p\}\}, \tag{4.17}$$

which is the optimal set of inputs contributing the upper bounds of their corresponding $\mathtt{MSR}$.

After computing $\mathsf{Solu}(\mathbb{X}, \mathcal{N}, t)_{\Delta}$, we only need one further network query to obtain all the upper bounds of the maximum safe radii for a given dataset $\mathbb{X}$.

Tightening the Upper Bounds

After computing the set of perturbed inputs contributing the upper bounds, i.e., $\mathbb{X}'$, we ask the question whether accumulating these adversarial perturbations along the row without missing any is necessary. The answer is "No". There might be redundancies in $\mathbb{X}' - \mathbb{X}$, i.e., not all the modifications in $\mathbb{X}' - \mathbb{X}$ are necessary to observe a misclassification. We can therefore remove the redundancies and thereby tighten the upper bounds. The procedure is more or less straightforward, as it is essentially a reverse process of Equation (4.16). From $\mathbb{X}'$, we iteratively remove perturbations in $\mathsf{Acc}[:, n_i, :]$ between the 1-th and the n_i-th rows to check if the misclassification is preserved. If yes, then the perturbation can be removed and the dimension restored to the original value, thus possibly tightening the upper bound; otherwise, the modification needs to be kept to retain the adversarial example. To sum up, we tighten the upper bounds by removing the possibly redundant adversarial perturbations in $\mathbb{X}'$, and therefore ensure the upper bound is strictly decreasing.

4.3.2 Anytime Robustness Evaluation

Until this point, we have developed algorithms to compute the lower and upper bounds of the maximum safe radii for a dataset. We remark that, for every input, the lower bound of $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d)$ is updated in an *increasing* manner, i.e., starting from $T = 1$, if all the solution points are safe then the lower bound increases accordingly until some adversarial example occurs. Meanwhile, the upper bound is generated from the accumulation of adversarial perturbations and then tightened via the restoration of the unnecessary modifications, i.e., the upper bound gradually *decreases*. Ideally, the increasing lower bound and the decreasing upper bound converge to the optimal value, i.e., $\mathtt{MSR}$. However, in practice the run times can be long when the input dimension is huge, i.e., large m, or a subspace contains many dimensions, i.e, t is large.

Regarding this, we propose an *anytime robustness evaluation* approach, providing pragmatic means to make progress. We first introduce the two sequences of the lower and the upper bounds.

Definition 4.7 (Sequences of Bounds)**.** *Given a verification of the local robustness problem* $\textsc{Robust}(\mathcal{N}, \mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d))$*, a sequence* $\mathsf{L}(\boldsymbol{\alpha}) = \{\mathsf{l}_1, \mathsf{l}_2, \ldots, \mathsf{l}_k\} \in \mathbb{R}$ *is an* incremental lower bound sequence *if, for all* $1 \leq i < j \leq k$*, we have* $\mathsf{l}_i \leq \mathsf{l}_j \leq \mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d)$*. The sequence is* strict*, denoted as* $\mathsf{L}_s(\boldsymbol{\alpha})$*, if for all* $1 \leq i < j \leq k$*, we have either* $\mathsf{l}_i < \mathsf{l}_j$ *or* $\mathsf{l}_i = \mathsf{l}_j = \mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d)$*. Similarly, we can define a* decremental upper bound sequence $\mathsf{U}(\boldsymbol{\alpha})$ *and a* strict *decremental upper bound sequence* $\mathsf{U}_s(\boldsymbol{\alpha})$*.*

Then we define the anytime local robustness evaluation as follows.

Definition 4.8 (Anytime Local Robustness Evaluation)**.** *Given a verification of local robustness problem* $\textsc{Robust}(\mathcal{N}, \mathsf{Ball}(\boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d))$*, compute a lower bound sequence* $\mathsf{L}(\boldsymbol{\alpha})$*, and an upper bound sequence* $\mathsf{U}(\boldsymbol{\alpha})$*, then at time* $t > 0$*,*

$$\mathsf{l}_t \leq \mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d) \leq \mathsf{u}_t \tag{4.18}$$

holds. Let its centre *be* $\frac{1}{2}(\mathsf{l}_t + \mathsf{u}_t)$ *and* radius *be* $\frac{1}{2}(\mathsf{u}_t - \mathsf{l}_t)$*, then the* anytime robustness evaluation of $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d)$ at time t *is the pair*

$$(\frac{1}{2}(\mathsf{l}_t + \mathsf{u}_t), \frac{1}{2}(\mathsf{u}_t - \mathsf{l}_t)). \tag{4.19}$$

The anytime evaluation will be returned whenever the computational procedure is interrupted. Intuitively, we use the centre $\frac{1}{2}(\mathsf{l}_t + \mathsf{u}_t)$ to represent the current *estimate*, and the radius $\frac{1}{2}(\mathsf{u}_t - \mathsf{l}_t)$ to represent its *error bound.* Essentially, we can bound the true maximum safe radius $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d)$ via the anytime robustness evaluation. Moreover, to extend to the anytime global robustness evaluation, we follow the computation of EMSR from MSR in Definition 4.3, and calculate the expected bounds as the mean of all the bounds for every input at time t, i.e.,

$$\mathbb{E}_{\alpha \in \mathbb{X}} \mathsf{l}_t \leq \texttt{EMSR}(\mathcal{N}, \mathbb{X}, \mathfrak{d}_{\mathsf{m}}, d) \leq \mathbb{E}_{\alpha \in \mathbb{X}} \mathsf{u}_t. \tag{4.20}$$

4.3.3 Convergence Analysis

We perform the convergence analysis of the proposed method. For simplicity, in the proofs we consider the case of a single input $\boldsymbol{\alpha}$. The convergence guarantee can be extended easily to a finite dataset $\mathbb{X}$.

Theorem 3 (Guarantee of Global Minimum of Grid Search)**.** *Assume that a neural network $\mathcal{N}$ with computation function $f(\boldsymbol{\alpha}) : [0,1]^m \to \mathbb{R}^n$ is Lipschitz continuous with respect to the L^p norm, and its Lipschitz constant is* Lip*. By recursively sampling $\Delta = 1/\epsilon$ in each dimension through grid search, denoted as $\mathsf{Subs}_\Delta = \{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{\Delta^m}\}$, the following relation holds:*

$$\left\| f_{\mathrm{opt}}(\boldsymbol{\alpha}^*) - \min_{\boldsymbol{\alpha} \in \mathsf{Subs}_\Delta} f(\boldsymbol{\alpha}) \right\|_p \leq \mathsf{Lip} \cdot \left\| \frac{\epsilon}{2} \mathbf{I}_m \right\|_p, \tag{4.21}$$

where $f_{\mathrm{opt}}(\boldsymbol{\alpha}^*)$ represents the global minimum *value,* $\min_{\boldsymbol{\alpha} \in \mathsf{Subs}_\Delta} f(\boldsymbol{\alpha})$ *denotes the minimum value returned by grid search, and $\mathbf{I}_m \in \mathbb{R}^{m \times m}$ is an all-ones matrix.*

Proof. Based on the Lipschitz continuity assumption of f, we have

$$\|f(\boldsymbol{\alpha}_1) - f(\boldsymbol{\alpha}_2)\|_p \leq \mathsf{Lip} \cdot \|\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2\|_p \tag{4.22}$$

The ϵ grid search guarantees that $\forall \tilde{\boldsymbol{\alpha}} \in [0,1]^m$, $\exists \boldsymbol{\alpha} \in \mathsf{Subs}_\Delta$ such that $\|\tilde{\boldsymbol{\alpha}} - \boldsymbol{\alpha}\|_p \leq \left\| \frac{\epsilon}{2} \mathbf{I}_m \right\|_p$. The global minimum $\boldsymbol{\alpha}^*$ must be some point in the input space $[0,1]^m$, therefore the theorem holds as we can always find $\boldsymbol{\alpha}$ from the sampled set Subs_Δ such that $\|\boldsymbol{\alpha}^* - \boldsymbol{\alpha}\|_p \leq \left\| \frac{\epsilon}{2} \mathbf{I}_m \right\|_p$. □

The guarantees for the lower and the upper bounds are explained as follows, with detailed proofs in [60].

Theorem 4 (Guarantee for Lower Bounds)**.** *Given a neural network $\mathcal{N}$, and an input $\boldsymbol{\alpha} \in [0,1]^m$, if our method generates a lower bound $\mathsf{l}(\mathcal{N}, \boldsymbol{\alpha})$, then $\mathcal{N}(\boldsymbol{\alpha}') = \mathcal{N}(\boldsymbol{\alpha})$ holds for all $\boldsymbol{\alpha}'$ such that $\mathfrak{d}_{\mathsf{m}}(\boldsymbol{\alpha}', \boldsymbol{\alpha}) \leq \mathsf{l}(\mathcal{N}, \boldsymbol{\alpha})$. That is, $\mathcal{N}$ is guaranteed to be safe for any pixel perturbations with at most $\mathsf{l}(\mathcal{N}, \boldsymbol{\alpha})$ pixels.*

Intuitively, Theorem 4 shows that the lower bounds generated by our algorithm are the lower bounds of $\texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d)$. We gradually increase $t = \mathsf{l}(\mathcal{N}, \boldsymbol{\alpha})$ and re-run the lower bound generation algorithm. Because the number of dimensions of an input is finite, the distance to an adversarial example is also finite. Therefore, the lower bound generation algorithm converges eventually.

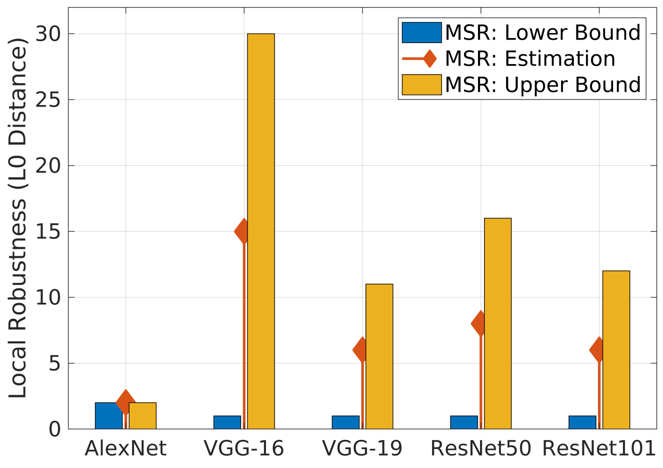
Theorem 5 (Guarantee for Upper Bounds)**.** *Given a neural network $\mathcal{N}$, and an input $\boldsymbol{\alpha} \in [0,1]^m$, if our method generates an upper bound $\mathsf{u}_t(\mathcal{N}, \boldsymbol{\alpha})$ for any $t > 0$, then $\mathsf{u}_{t+1}(\mathcal{N}, \boldsymbol{\alpha}) \leq \mathsf{u}_t(\mathcal{N}, \boldsymbol{\alpha})$ holds for all $t > 0$, and $\lim_{t \mapsto \infty} \mathsf{u}_t(\mathcal{N}, \boldsymbol{\alpha}) = \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, \mathfrak{d}_{\mathsf{m}}, d)$.*

Intuitively, there are three key ingredients to prove the monotonic decrease of the upper bounds: (1) the complete subspaces generated at t are always included in the complete subspaces at $t+1$; (2) the pixel perturbation from a subspace with higher sensitivity always results in a larger confidence decrease than those with lower sensitivity; and (3) the tightening strategy is able to exclude the redundant pixel perturbations.

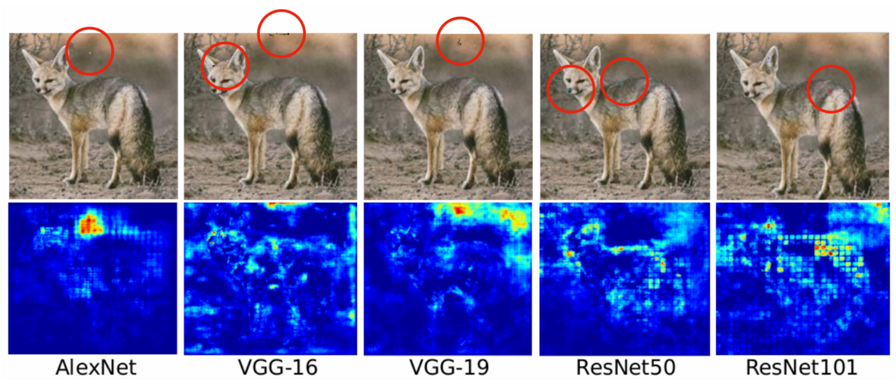
4.4 Experimental Results

We implement our approach in a tool named DeepTRE[2], short for "Tensor-based Robustness Evaluation for Deep Neural Networks", and report experimental evidence for the utility of our algorithm. Some experiments may require simple adaptations of the optimisation problem in Definition 4.3, e.g., small changes to the constraints, nevertheless, no significant modification to the algorithm itself is needed to process

[2] Available on GitHub: `https://github.com/TrustAI/L0-TRE`



(a) Lower/upper bounds and estimates of MSR



(b) Adversarial examples (top) and saliency maps (bottom)

Figure 4.3: Local robustness and subspace sensitivity of common ImageNet networks. (a) Lower bounds, upper bounds, and estimates of the local robustness for the five neural networks with respect to an ImageNet image. (b) Adversarial examples on the upper boundaries with perturbations in red circle (top) and the corresponding saliency maps where a brighter colour indicates a more sensitive pixel (bottom).

these variants. In this section, we use several case studies to demonstrate the broad applicability of DeepTRE.

4.4.1 Saliency Maps and Local Robustness

We study the relation of subspace sensitivity and local robustness via applying our method to five state-of-the-art network models trained on the ImageNet dataset, including AlexNet (8 layers), VGG-16 (16 layers), VGG-19 (19 layers), ResNet50 (50 layers), and ResNet101 (101 layers). Introduction to these ImageNet models, as well as the details of the experimental setup, can be found in Appendix A.1.

In this case, we set the subspace dimension $t = 1$, which means that we consider the sensitivity of each pixel individually, i.e., not including groups of different pixels. Based on this single pixel level sensitivity, in Figure 4.3(b) we generate the saliency maps of the five ImageNet models with respect to the same "kit fox" image. We observe that, while AlexNet and VGG-19 somehow "think" the background is more salient, VGG-16 and ResNet50/101 "see" more into the silhouette of the image. Correspondingly, based on the saliency map, we produce an adversarial example for each model, from which we can see that the adversarial perturbations for VGG-16 and ResNet50/101 are basically imposed on the face and the back of the "kit fox". In other words, this is closer to where humans would focus on when we try to distinguish the animal in the image, not just some random place in the background.

Nevertheless, merely analysing a network's adversarial examples is not enough. We measure the robustness quantitatively via computing the MSR for each model. As the ImageNet dataset is large in dimensions thus making it difficult to compute the MSR, we compute a lower and upper bound for it and give the mean as an estimate. Figure 4.3(a) reveals that VGG-16 and ResNet50/101 have a relatively high MSR, i.e., they are comparatively more robust with respect to this specific image. This result is in fact consistent with the phenomena observed from saliency maps. Moreover, if we compare similar network structures, such as VGG-16/19 and ResNet50/101, we may conclude that a model with deeper layers is not necessarily more robust to adversarial perturbations, as VGG-19 and ResNet101 have a lower MSR.

Specifically, we remark that for AlexNet with respect to this specific image, the lower and upper bounds converge to the actual MSR, i.e., $\mathfrak{d}_{\mathsf{m}} = 2$. Intuitively, it means that manipulating a single arbitrary pixel of the "kit fox" image will not alter the classification of AlexNet, whereas changing not more than two pixels will definitely result in an adversarial example. That said, DeepTRE is able to find the ground-truth adversarial example[3] for AlexNet on this image.

[3]Ground-truth adversarial images are images at the boundary of a safe metric ball, which were

4.4.2 Convergence of Bounds and Global Robustness

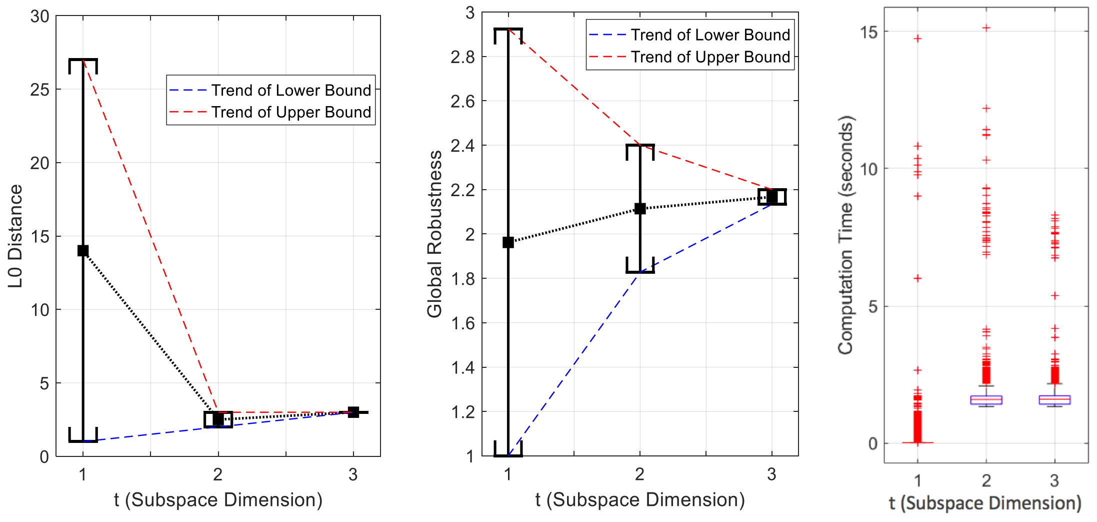
Now we generalise from searching for adversarial examples to analysing local and global robustness of neural networks. Recall from Theorem 2 that computing the MSR directly is NP-hard, and the complexity is determined primarily by the *number of input dimensions*. Therefore, in this section we evaluate robustness through demonstrating convergence of the lower and upper bounds, and also examine how input dimension would affect convergence.

Specifically, we train two neural networks on the MNIST dataset with different input dimensions, namely DNN-Reduced and DNN-Standard. The former is trained on the dataset with reduced image size 14×14, whereas the latter trained with original size of 28×28. To compute global robustness, i.e., EMSR, for DNN-Reduced we work with a set of 5300 randomly sampled test images, and for DNN-Standard we use a set of 2400 test images. The structure of these two networks and their training/testing accuracy statistics can be found in Appendix A.2, together with the parameter settings of DeepTRE and the hardware/software platforms.

DNN-Reduced: Local/Global Robustness and Convergence of Bounds

We first look into local robustness, which is essentially a special case of global robustness where $|\mathbb{X}| = 1$, i.e., on a single image instead of a dataset. Figure 4.4(a) illustrates the converging lower and upper bounds of the MSR of DNN-Reduced with subspace dimension $t \in \{1, 2, 3\}$. We choose this particular image to demonstrate the worst case of our approach – when the subspace dimension is one pixel, the initial upper bound $\mathfrak{d}_{\mathsf{m}} = 27$ is actually quite large. However, after increasing the subspace dimension to two pixels, the upper bound decreases drastically from 27 to 3 and simultaneously the lower bound increases from 1 to 2. That is, the uncertainty radius of MSR is reduced significantly from 26 to 1. Furthermore, when the subspace dimension is three pixels, the bounds converge to the actual MSR $= 3$. Intuitively, we

proposed in [7].



(a) Local robustness (b) Global robustness (c) Time

Figure 4.4: Local and global robustness of DNN-Reduced with subspace dimension $t \in \{1, 2, 3\}$. (a) Convergence of lower and upper bounds, together with the estimate of `MSR` for one image. (b) Convergence of lower and upper bounds, together with the estimate of `EMSR` for the test dataset. (c) Box-plots of computational time with respect to subspace dimension.

can say that if modifying this MNIST image with fewer than three pixels, let it be any location or combination of pixels, the classification of RNN-Reduced will not alter.

Now we extend to global robustness properly over the test dataset when $|\mathbb{X}| = 5300$ in Figure 4.4(b). In general, we observe that our approach obtains tight lower and upper bounds efficiently and these bounds converge quickly. Notably, we have the estimate of `EMSR` as 1.97 when the subspace dimension is one pixel, and after adding another pixel of subspace, the estimate becomes 2.1, so the relative error is less than 7%. Moreover, the relative error is even smaller when the subspace dimension changes to three pixels. In other words, our method is able to provide a good approximation of `EMSR` with reasonable error when the computation proceeds, until eventually the actual `EMSR` is obtained when the bounds converge. Concerning convergence, in Figure 4.5 we plot the convergence condition for all these 5300 test images. These charts show a clear overall trend that our algorithm converges for most images after a few iterations.

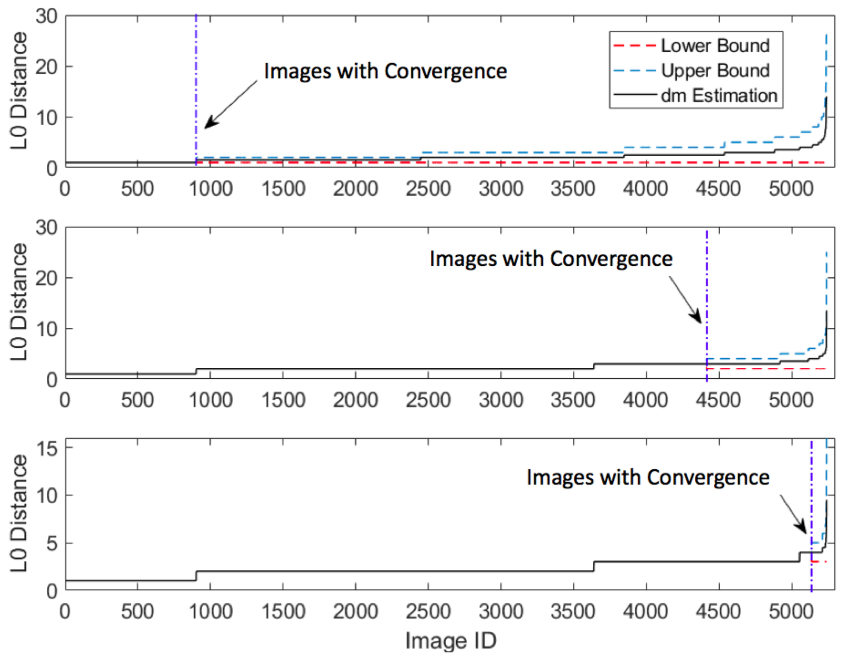
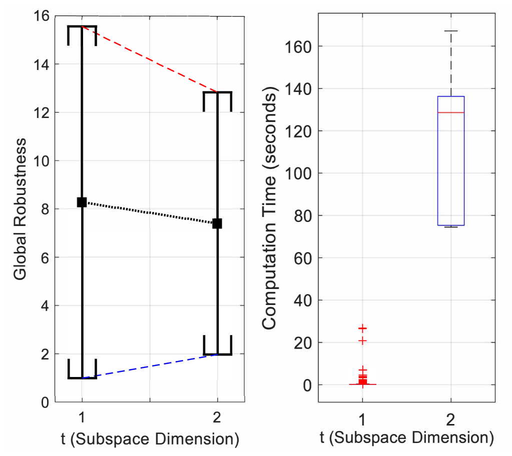


Figure 4.5: Convergence of lower and upper bounds of DNN-Reduced on the test dataset. From top to bottom: convergence of all sampled test images for subspace dimension $t \in \{1, 2, 3\}$, respectively. The vertical blue line indicates that all images to the left have converged.

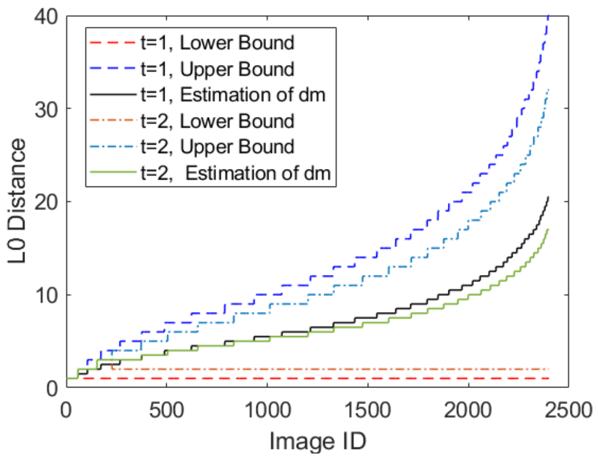
As for the computational time, Figure 4.4(c) gives the box-plots of the cost for local robustness with respect to different subspace dimensions. We remark that, when the subspace is one pixel, our approach takes less than 0.1 s to process an image, which suggests its potential application for real-time scenarios.

DNN-Standard: Scalability of Robustness and Convergence

We evaluate the scalability of our robustness evaluation via convergence of bounds over different input dimensions. Regarding this, the model DNN-Standard is trained on the original MNIST training set, and we present its robustness and convergence for 2400 test images in Figure 4.6. We observe that, even for a network with tens of thousands of hidden neurons, DeepTRE can still achieve tight estimates of `EMSR` for most images, though the lower and upper bounds require more computational resource to converge to the actual values, due to the increase of the image size. Also, the large size of the input affects the execution time because, when the



(a) Global robustness and time



(b) Lower and upper bounds

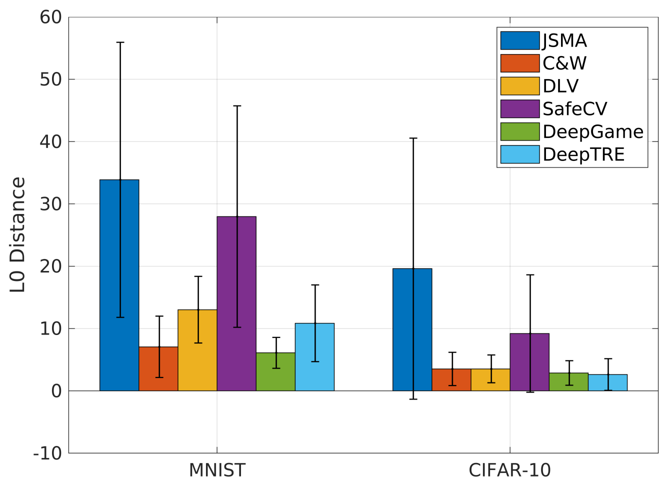
Figure 4.6: Global robustness and convergence of DNN-Standard on the test dataset with subspace dimension $t \in \{1, 2\}$. (a) Global robustness of DNN-Standard and box-plots of computational time. (b) Lower and upper bounds, together with the estimates of MSR for all sampled test images.

subspace dimension changes from one to two pixels, the choice of subspace increases considerably by $\binom{2}{28\times 28}$.

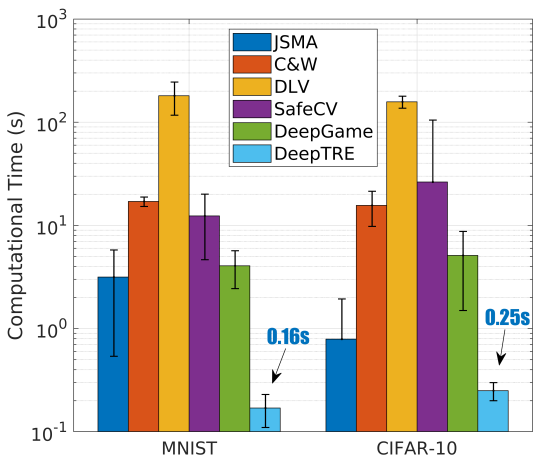
Besides, if we compare global robustness of DNN-Standard and DNN-Reduced, we discover that the former has a higher EMSR (at least when $t \in \{1, 2\}$), which may be because DNN-Standard is trained on the original, rather than the reduced size version of the MNIST dataset, and therefore it tends to be more robust against adversarial perturbations. Finally, we exhibit some ground-truth adversarial images returned by DeepTRE in Figure A.2.4 in the Appendix.

4.4.3 Competitive Adversarial Attacks

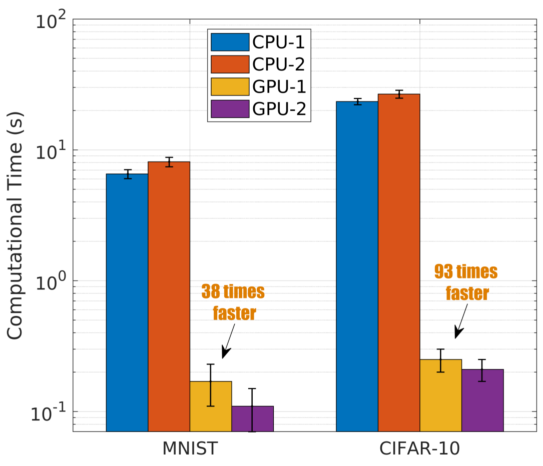
While generating adversarial examples is not the primary goal of our approach, we remark that our upper bound algorithm is highly competitive with the state-of-the-art adversarial attack methods in terms of the Hamming distance. Specifically, we train two MNIST and CIFAR-10 networks and compare on 1000 test images with other tools, such as JSMA [53], C&W [8], DLV [28], SafeCV [76], and DeepGame [80].



(a) Hamming distance



(b) Computational time



(c) CPUs vs. GPUs

Figure 4.7: Competitive adversarial attacks of DeepTRE and other tools. (a) Means and standard deviations of the adversarial Hamming distance. (b) Means and standard deviations of the computational time of all tools. (c) Means and standard deviations of the computational time of DeepTRE with CPUs or GPUs.

Model structure and training/testing accuracy, as well as the parameter setting of each tool, are given in Appendix B.2.

Adversarial Hamming Distance

We evaluate the efficiency of adversarial attacks in terms of the Hamming distance to the original image and the computational time. Intuitively, a small adversarial distance indicates a more subtle perturbation, which is thus less likely to be perceived by humans. Figure 4.7(a) depicts the means and standard deviations of the Hamming distances of the adversarial images produced by these six tools. We observe that, on the MNIST dataset, our tool DeepTRE performs better than JSMA, DLV, and SafeCV, and is comparable to C&W and DeepGame. As for CIFAR-10, the bar chart reveals that DeepTRE achieves the smallest adversarial distance among all competitors, modifying only 2.62 pixels on average.

Computational Cost

As for the computational time, we compare all these tools against DeepTRE when using CPUs and GPUs. The latter is to show the advantage of our tensor-based parallelisation during the computation process. Figure 4.7(b) presents the running times of all these tools. Note that the vertical axis is in logarithmic scale. We can see that, for both MNIST and CIFAR-10 networks, our tool delivers very efficient adversarial attacks. For example, on the MNIST model, DeepTRE performs 18×, 100×, 1050×, 357×, and 23× faster than JSMA, C&W, DLV, SafeCV, and DeepGame, respectively. Moreover, Figure 4.7(c) demonstrates that the utilisation of tensor-based parallelisation significantly improves computational efficiency. For instance, when using GPUs, DeepTRE is 38× faster in attacking the MNIST model and 93× faster on CIFAR-10. Below we list the hardware setups:

- "CPU-1" – Tensorflow on an i5-4690S CPU;
- "CPU-2" – Deep Learning Toolbox (MATLAB 2018b) on an i7-7700HQ CPU;
- "GPU-1" – Tensorflow with parallelisation on an NVIDIA GTX TITAN GPU;
- "GPU-2" – Deep Learning Toolbox (MATLAB 2018b) with parallelisation on an NVIDIA GTX-1050Ti GPU.

Finally, we compare some adversarial examples of the same images found by these six tools in Figures B.1 and B.2 of Appendix B.2. These attack methods indicate that modifying one to three pixels suffices to trigger a misclassification of a well-trained neural network.

Table 4.1: A sketch of the architectures of the MNIST models from DNN-1 to DNN-7 together with their accuracy rates on the test dataset. Each integer indicates the number of layer types, activation functions, or operations. Detailed model structures are given in Appendix A.3.2.

	DNN-1	DNN-2	DNN-3	DNN-4	DNN-5	DNN-6	DNN-7
Convolutional	1	2	2	2	2	3	4
ReLU	1	2	2	3	3	4	5
Batch-Normalisation			1	1	1	1	3
Dropout					1	1	2
Fully Connected	1	1	1	2	2	2	2
Accuracy (%)	97.75	97.95	98.38	99.06	99.16	99.13	99.41

4.4.4 Robustness and Accuracy of Model Architectures

In this section, we examine how different model architectures of a neural network affect its accuracy rates on the same training and test datasets, as well as its robustness against adversarial perturbations. A general intuition is that a network with deeper layers tends to be more accurate and thus is more robust. Below, we perform experiments to demonstrate that this is not necessarily true.

Specifically, we train seven different networks on the MNIST dataset, namely DNN-i for $\mathsf{i} \in \{1, \ldots, 7\}$. These networks vary in model architectures in terms of the depth of layers and also the number of layer types, activation functions, or operations. While the detailed structures are listed in Appendix A.3.2, we provide a sketch of them in Table 4.1, where the complexity gradually increases from DNN-1 to DNN-7. All models are trained with identical training parameters on the same hardware and software platforms, attached in Appendix A.3. Regarding the accuracy rates, while all these networks achieve 100% accuracy on the training set, we observe that the testing accuracy increases generally from 97.75% to 99.41%.

We evaluate global robustness of these seven networks on 1000 sampled test images. For each network, Figure 4.8 plots the expected maximum safe radius (EMSR)

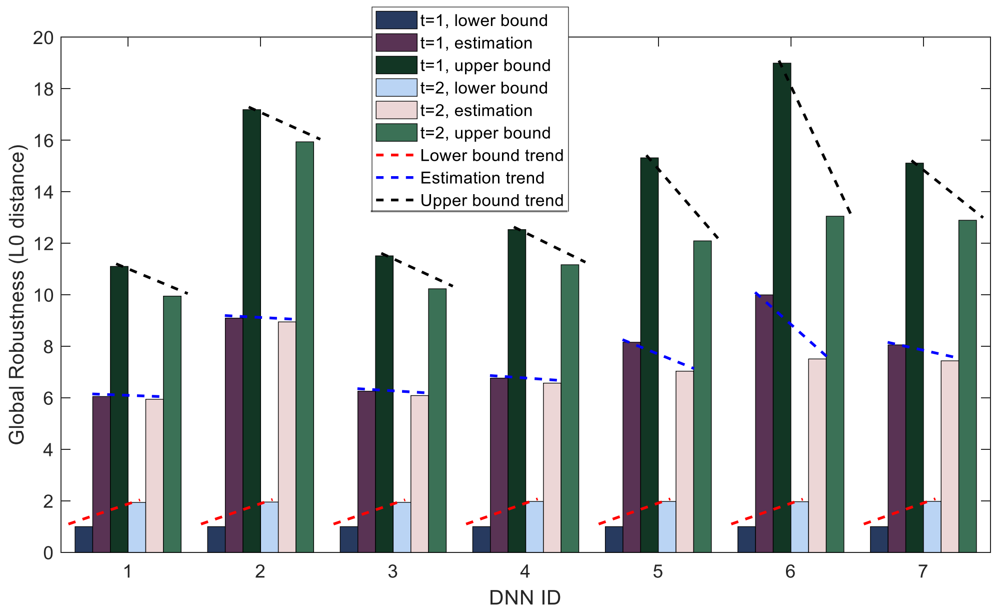


Figure 4.8: Global robustness in terms of lower and upper bounds of the *expected maximum safe radius* (`EMSR`) of seven MNIST networks with varying model structures on 1000 test images when the subspace dimension $t \in \{1, 2\}$.

and its lower and upper bounds with respect to subspace dimension $t \in \{1, 2\}$. By comparing `EMSR` of all the models, we endeavour to identify the architectural choices that would affect robustness, in this case, against adversarial perturbations based on the Hamming distance. Recall that a greater `EMSR` indicates better robustness. We observe the following:

- *depth of network*: A deeper network, at least with too many layers relative to the dimension of each input in the dataset, is not necessarily more robust. For example, here DNN-7 is the deepest and has the most complicated structure, but is in fact not as robust as DNN-6.
- *convolutional layer*: A convolutional layer with ReLU as the activation function is likely to improve the robustness of a network. From the figure, we discover that the `EMSR` increases from DNN-3 to DNN-6, accompanied by the addition of more convolutions and ReLU functions in the table. This is also reflected by DNN-1 and DNN-2, where the latter has an extra convolutional layer with ReLU and is more robust.

- *batch-normalisation*: A batch-normalisation operation, though it enhances accuracy, might actually compromise the robustness of a network. In the figure, we spot an obvious drop of EMSR from DNN-2 to DNN-3 and also from DNN-6 to DNN-7, which is probably caused by the existence of batch-normalisation in DNN-3 and the extra two batch-normalisation operations in DNN-7.

To this end, we remark that accuracy rate measured on the training and test datasets is not necessarily a good proxy for robustness: a network with higher accuracy might turn out to be less robust. For example, in our experiments DNN-7 is less robust than DNN-2 while the former has a remarkably higher accuracy rate. That said, neural networks may require a balance between robustness and their ability to generalise, which is currently represented by the testing accuracy [69].

4.5 Summary

To the best of our knowledge, this is the *first* algorithm that evaluates local and global robustness of deep neural networks with provable guarantees based on the Hamming distance. We provide a tensor-based implementation of the technique to exploit the inherent parallelism. Our experimental results demonstrate wide applicability and efficiency of the method, with potential for real-time deployment. We hypothesise that the approach computes a good proxy for robustness against physical attacks that rely on the manipulation of a small part of the objects that are to be recognised.

5
Robustness of Deep Neural Networks on Features of An Image

Contents

In this chapter, apart from the maximum safe radius considered in the previous chapter, we also study the *feature robustness* problem, which aims to find a feature, or a subset of features on a given input, that is the most robust against adversarial perturbations. To evaluate the robustness on the feature-level is partially inspired by the sensitivity analysis and the saliency maps in Chapter 4. It is surprising to see that a well-trained network would identify an image into a specific class, not because of the pixels within the main object, but due to some rather "random" pixels (at least to human perception) in the background. This does not make sense because, when humans perceive an image, we immediately identify the salient features and almost instantly exclude the irrelevant elements out of subconsciousness. Regarding this, we think feature robustness is an exciting problem to dig into.

As for the metrics, instead of the Hamming distance, in this chapter we work with the L^p norms, i.e., the Manhattan distance (L^1 norm), the Euclidean distance (L^2 norm), and the Chebyshev distance (L^∞ norm), in order to study other possible physically plausible distortions such as "brightness change". Recall that, in Section 3.2, we remark that the Hamming distance is not actually a norm as it does not satisfy positive homogeneity (Remark 1). Based on this, we highlight the key differences between the approach in Chapter 4 and this chapter, and explain in two respects why they cannot be applied on these metrics interchangeably.

- On one hand, for the pixel-based approach in Chapter 4, because the number of input dimensions is finite, the Hamming distance to an adversarial example is always finite, i.e., $\mathfrak{d}_{\mathsf{m}} \in \mathbb{N}^+$. Moreover, no matter how a single dimension is perturbed, the resulting adversarial distance always remains as one pixel. However, when dealing with the L^p norms, where the distance values are not integers but real numbers, i.e., $\|\boldsymbol{x} - \boldsymbol{x}'\|_p \in \mathbb{R}^+$, the pixel-based approach is no longer practically applicable as the adversarial distance is now dependent on each small perturbation in every pixel.
- On the other hand, the method proposed in this chapter guarantees robustness based on the L^p norms but cannot generalise the guarantees to the Hamming

distance. This is because in this chapter we utilise Lipschitz continuity to discretise the input space so that the two problems, the maximum safe radius and feature robustness, are reduced to finite optimisations with provable guarantees. However, from Definition 3.5 we know that $\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|_p \leq \mathsf{Lip} \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|_p$ no longer holds for the Hamming distance. If we relax the constraints on guarantees, e.g., focus on generating adversarial examples instead, then this method also works with the Hamming distance, which is demonstrated in Section 5.4.3.

The organisation of this chapter is as follows. In Section 5.1, we first describe the two problems, maximum safe radius and feature robustness, and then reduce them into two finite optimisations while showing that such reductions have provable guarantees under the assumption of Lipschitz continuity. Then, in Section 5.2, we show that the finite optimisation problems can be computed as the solution of two-player turn-based games, where Player `I` selects features and Player `II` then performs atomic perturbations within the chosen features. We then exploit algorithms to compute the upper and the lower bounds of Player `I`'s reward, and provide convergence analysis in Section 5.3. Finally, we implement the algorithms into a tool DeepGame and report the experimental results in Section 5.4.

5.1 Robustness on Features of An Image

Recall that we have defined the metric ball Ball and the maximum safe radius `MSR` in the previous chapter, here we continue working with these two concepts, and consider another problem – the *feature robustness* problem, which we introduce in Section 5.1.2. Besides, based on adversarial examples (Definition 3.7), we extend it to the notions of *targeted* and *non-targeted safety* as follows.

Definition 5.1 (Targeted and Non-Targeted Safety). *Given an input $\boldsymbol{\alpha} \in \mathrm{D}$, a distance measure L^p for some $p \geq 0$, and a distance d, an* adversarial example *$\boldsymbol{\alpha}'$ of class c is such that $\boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$, $\mathcal{N}(\boldsymbol{\alpha}) \neq \mathcal{N}(\boldsymbol{\alpha}')$, and $\mathcal{N}(\boldsymbol{\alpha}') = c$.*

Moreover, we write $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$ *for the set of adversarial examples of class* c *and let*

$$\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}) = \bigcup_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha})} \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c). \tag{5.1}$$

A targeted safety *of class* c *is defined as* $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c) = \emptyset$*, and a* non-targeted safety *is* $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}) = \emptyset$*.*

The following formalisation focuses on the targeted safety of a network $\mathcal{N}$ with respect to a given input $\boldsymbol{\alpha}$ and a fixed class $c \neq \mathcal{N}(\boldsymbol{\alpha})$. We remark that the situation of non-targeted safety, i.e., misclassification into class other than c, is similar.

Input Manipulations

To study the crafting of adversarial examples, we require the following operations for manipulating inputs. Let $\tau > 0$ be a positive real number representing the manipulation magnitude, then we can define the *input manipulation* operations $\delta_{\tau,X,\psi} : \mathrm{D} \to \mathrm{D}$ for $X \subseteq P_0$, a subset of input dimensions, and $\psi : P_0 \to \mathbb{N}$, an instruction function by:

$$\delta_{\tau,X,\psi}(\boldsymbol{\alpha})(i) = \begin{cases} \boldsymbol{\alpha}[i] + \psi(i) * \tau, & \text{if } i \in X \\ \boldsymbol{\alpha}[i], & \text{otherwise} \end{cases} \tag{5.2}$$

for all $i \in P_0$. Note that if the values are bounded, e.g., in the interval $[0, 1]$, then $\delta_{\tau,X,\psi}(\boldsymbol{\alpha})(i)$ needs to be restricted to be within the bounds. Let Ψ be the set of possible instruction functions.

The following lemma shows that input manipulation operations allow one to map one input to another by changing the values of input dimensions, regardless of the distance measure L^p.

Lemma 1. *Given any two inputs* $\boldsymbol{\alpha}$ *and* $\boldsymbol{\alpha}'$*, and a distance* $\|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_p$ *for any measure* L^p*, there exists a magnitude* $\tau > 0$*, an instruction function* $\psi \in \Psi$*, and a subset* $X \subseteq P_0$ *of input dimensions, such that*

$$\|\boldsymbol{\alpha} - \delta_{\tau,X,\psi}(\boldsymbol{\alpha}')\|_p \leq \epsilon \tag{5.3}$$

where $\epsilon > 0$ *is an error bound.*

Intuitively, any distance can be implemented through an input manipulation with an error bound ϵ. The error bound ϵ is needed because input $\alpha \in \mathrm{D} = [0,1]^n$ is bounded, and thus reaching another precise input point via a manipulation is difficult when each input dimension is a real number.

We will also distinguish a subset of *atomic* input manipulations, each of which changes a single dimension for a single magnitude.

Definition 5.2. *Given a set X, we let $\Delta(X)$ be the set of* atomic *input manipulations δ_{τ,X_1,ψ_1} such that*

- $X_1 \subseteq X$ *and* $|X_1| = 1$, *and*
- $\psi_1(i) \in \{-1,+1\}$ *for all* $i \in P_0$.

Lemma 2. *Any input manipulation $\delta_{\tau,X,\psi}(\boldsymbol{\alpha})$ for some X and ψ can be implemented with a finite sequence of input manipulations*

$$\delta_{\tau,X_1,\psi_1}(\boldsymbol{\alpha}), \ldots, \delta_{\tau,X_m,\psi_m}(\boldsymbol{\alpha}) \in \Delta(X). \tag{5.4}$$

While the existence of a sequence of atomic manipulations implementing a given manipulation is determined, there may exist multiple sequences. On the other hand, from a given sequence of atomic manipulations we can construct a single input manipulation by sequentially applying the atomic manipulations.

Feature-Based Partitioning

As introduced in Section 3.4.2, natural data forms a high-dimensional manifold, and feature manifolds usually have lower dimensions than the data manifold. In other words, the set of features form a partition of the input dimensions. Formally, we define the feature-based partition of an input as follows.

Definition 5.3 (Feature Extraction)**.** *Let λ be a feature of an input $\boldsymbol{\alpha} \in \mathrm{D}$, then we use $P_\lambda \subseteq P_0$ to denote the dimensions represented by λ. Given an input $\boldsymbol{\alpha}$, a* feature extraction *function Λ maps an input $\boldsymbol{\alpha}$ into a set of features $\Lambda(\boldsymbol{\alpha})$ such that (1) $P_0 = \bigcup_{\lambda \in \Lambda(\boldsymbol{\alpha})} P_\lambda$, and (2) $P_{\lambda_i} \cap P_{\lambda_j} = \emptyset$ for any*

$\lambda_i, \lambda_j \in \Lambda(\boldsymbol{\alpha})$ *with* $i \neq j$.

In this chapter, we employ two feature extraction approaches to partition an input into disjoint subsets, namely the *saliency*-guided grey-box and the *feature*-guided black-box methods. The former utilises the saliency map generated from the subspace sensitivity in Chapter 4, whereas the latter applies the SIFT algorithm [41] to the input locally. Section 5.4.1 provides an experimental illustration of these methods.

We remark that our technique is not limited to image classifiers and is able to work with general classification tasks, as long as there is a suitable feature extraction method that generates a partition of the input dimensions. In this case, we focus on image classification for illustrative purpose and to enable better comparison.

5.1.1 The Maximum Safe Radius Problem

Given a targeted safety problem for $\boldsymbol{\alpha}$, we aim to compute the distance $\|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_p$ to the nearest adversarial example within the d-neighbourhood of $\boldsymbol{\alpha}$, or in other words the radius of the maximum safe metric ball, illustrated in Figure 4.1. Note that the definition below is very similar to Definition 4.2, with the only difference being the incorporation of a class c, which denotes a targeted safety problem. We keep it here to avoid confusion.

Definition 5.4 (Maximum Safe Radius)**.** *The* maximum safe radius *problem is to compute the minimum distance from the original input* $\boldsymbol{\alpha}$ *to an adversarial example, i.e.,*

$$\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) = \min_{\boldsymbol{\alpha}' \in \mathrm{D}} \{\|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_p \mid \boldsymbol{\alpha}' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)\} \tag{5.5}$$

If $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c) = \emptyset$*, we let* $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) = d + \epsilon$.

Intuitively, $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ represents an *absolute* safety radius within which all inputs are safe. In other words, within a distance less than $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$, no adversarial example is possible. When no adversarial example can be found within radius d, i.e., $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c) = \emptyset$, the maximum safe radius cannot be computed, but is definitely greater than d. Therefore, we let $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) = d + \epsilon$.

Recall that finding an adversarial example can only provide a loose upper bound of `MSR`. Similarly, this chapter investigates the more fundamental problem – how to approximate the *true* `MSR` distance with provable guarantees.

Approximation Based on Finite Optimisation

Note that the two adversarial sets $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$ and $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha})$ can be infinite. We now present a discretisation method that allows us to approximate the maximum safe radius using finite optimisation, and show that such a reduction has provable guarantees, provided that the network is Lipschitz continuous. Our approach proceeds by constructing a finite 'grid' of points in the input space. Lipschitz continuity enables us to reduce the verification problem to manipulating just the grid points, through which we can bound the output behaviour of a network on the whole input space, since Lipschitz continuity ensures that the network behaves well within each cell. The number of grid points is proportional to the Lipschitz constant. However, estimating a tight Lipschitz constant is difficult, and so, rather than working with the Lipschitz constant directly, we assume the existence of a (not necessarily tight) Lipschitz constant and work instead with a chosen fixed magnitude of an input manipulation, i.e., $\tau \in (0, 1]$. We show how to determine the largest τ for a given Lipschitz network and give error bounds for the computation of `MSR` that depend on τ.

We begin by constructing, for a chosen fixed magnitude $\tau \in (0, 1]$, the input manipulations to search for adversarial examples.

Definition 5.5 (Finite Maximum Safe Radius). *Let $\tau \in (0, 1]$ be a manipulation magnitude. The* finite maximum safe radius *problem* $\mathtt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ *based on input manipulation is as follows:*

$$\min_{\Lambda' \subseteq \Lambda(\boldsymbol{\alpha})} \min_{X \subseteq \bigcup_{\lambda \in \Lambda'} P_\lambda} \min_{\psi \in \Psi} \{ \|\boldsymbol{\alpha} - \delta_{\tau,X,\psi}(\boldsymbol{\alpha})\|_p \mid \delta_{\tau,X,\psi}(\boldsymbol{\alpha}) \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c) \}. \quad (5.6)$$

If $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c) = \emptyset$*, we let* $\mathtt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) = d + \epsilon$*.*

Intuitively, we aim to find a set Λ' of features, a set X of dimensions within Λ', and a manipulation instruction ψ such that the application of the atomic manipulation $\delta_{\tau,X,\psi}$ on the original input $\boldsymbol{\alpha}$ leads to an adversarial example $\delta_{\tau,X,\psi}(\boldsymbol{\alpha})$ that is nearest to $\boldsymbol{\alpha}$ among all adversarial examples. Compared to Definition 5.4, the search for another input by $\min_{\boldsymbol{\alpha}' \in \mathrm{D}}$ over an infinite set is implemented by minimisation over the finite sets of feature sets and instructions.

To this point, we have reduced the (infinite) maximum safe radius (`MSR`) problem into the discretised finite maximum safe radius (`FMSR`) problem, as in $\mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$ there are potentially infinitely many $\boldsymbol{\alpha}'$, whilst the number of $\delta_{\tau,X,\psi}(\boldsymbol{\alpha})$ is finite. Next, we demonstrate that we can use `FMSR` to bound the value of `MSR` with provable guarantees. Intuitively, from Definitions 5.4 and 5.5, we know that `MSR` denotes the precise minimum distance from $\boldsymbol{\alpha}'$ to $\boldsymbol{\alpha}$, while `FMSR` finds the nearest $\delta_{\tau,X,\psi}(\boldsymbol{\alpha})$ with the minimum atomic manipulations. And since the manipulations are finite, it follows naturally that `MSR` is not greater than `FMSR`. Therefore, we have the following lemma.

Lemma 3. *For any manipulation magnitude $\tau \in (0, 1]$, we have that*

$$\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) \leq \mathtt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau). \tag{5.7}$$

To ensure the other direction of $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ in Lemma 3, we utilise the fact that the network is Lipschitz continuous [59]. First, we need the concept of *a τ-grid input* for a manipulation magnitude τ. The intuition for the τ-grid is illustrated in Figure 5.1. We construct a finite set of grid points uniformly spaced by τ in such a way that they can be covered by small subspaces centred on grid points. We select a sufficiently small value for τ based on a given Lipschitz constant so that all points in these subspaces are classified the same. We then show that an optimum point (e.g., `FMSR`) on the grid is within an error bound dependent on τ from the true optimum, i.e., the closest adversarial example (e.g., `MSR`).

Definition 5.6 (τ-Grid Input)**.** *An input* $\boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$ *is* a τ-grid input *if, for all dimensions* $p \in P_0$*, we have* $|\boldsymbol{\alpha}'(p) - \boldsymbol{\alpha}(p)| = n \times \tau$ *for some* $n \geq 0$*. Let* $\Gamma(\boldsymbol{\alpha}, L^p, d, \tau)$ *be the* set of τ-grid inputs *in* $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$*.*

We note that τ-grid inputs in the set $\Gamma(\boldsymbol{\alpha}, L^p, d, \tau)$ are reachable from each other by applying an input manipulation. The main purpose of defining the τ-grid inputs is to ensure that the space $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$ can be covered by small subspaces centred on the grid points. To implement this, we need the following lemma. The proof is technical and given in Appendix B.1.

Lemma 4. *We have* $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d) \subseteq \bigcup_{\boldsymbol{\alpha}' \in \Gamma(\boldsymbol{\alpha}, L^p, d, \tau)} \mathsf{Ball}(\boldsymbol{\alpha}', L^p, \frac{1}{2} d(L^p, \tau))$*, where* $d(L^p, \tau) = (|P_0| \tau^p)^{\frac{1}{p}}$ *is the grid width.*

Intuitively, this means that the metric ball of $\boldsymbol{\alpha}$ is discretised into a finite grid of inputs, and each grid point $\boldsymbol{\alpha}'$ has its own metric ball with radius $\frac{1}{2} d(L^p, \tau)$ covering a small subspace. The key point is that the union of all these small subspaces should cover the metric ball of the original input. Also, because each grid point has radius $\frac{1}{2} d(L^p, \tau)$, the distance between two neighbouring grid points is then $d(L^p, \tau)$, which we call the 'grid width' here. As shown in Figure 5.1, the distance $\frac{1}{2} d(L^p, \tau)$ is the radius of the norm ball subspaces covering the input space. It is not difficult to see that, for different norms, $d(L^1, \tau) = |P_0| \tau$, $d(L^2, \tau) = \sqrt{|P_0| \tau^2}$, and $d(L^\infty, \tau) = \tau$.

We have thus discretised the whole input space into a union of the small subspaces centred on the grid points. In order to reduce the need to consider all the infinite points to manipulating just the grid points, we need to make sure that the network behaves well in each subspace (or grid cell). In other words, a grid point should be able to 'represent' all the other inputs in its metric ball. To realise this, we need the concept of a *misclassification aggregator*.

Definition 5.7 (Misclassification Aggregator)**.** *An input* $\boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$ *is a* misclassification aggregator *with respect to a number* $\beta > 0$ *if, for any* $\boldsymbol{\alpha}'' \in \mathsf{Ball}(\boldsymbol{\alpha}', L^p, \beta)$*, we have that* $\mathcal{N}(\boldsymbol{\alpha}'') \neq \mathcal{N}(\boldsymbol{\alpha})$ *implies* $\mathcal{N}(\boldsymbol{\alpha}') \neq \mathcal{N}(\boldsymbol{\alpha})$*.*

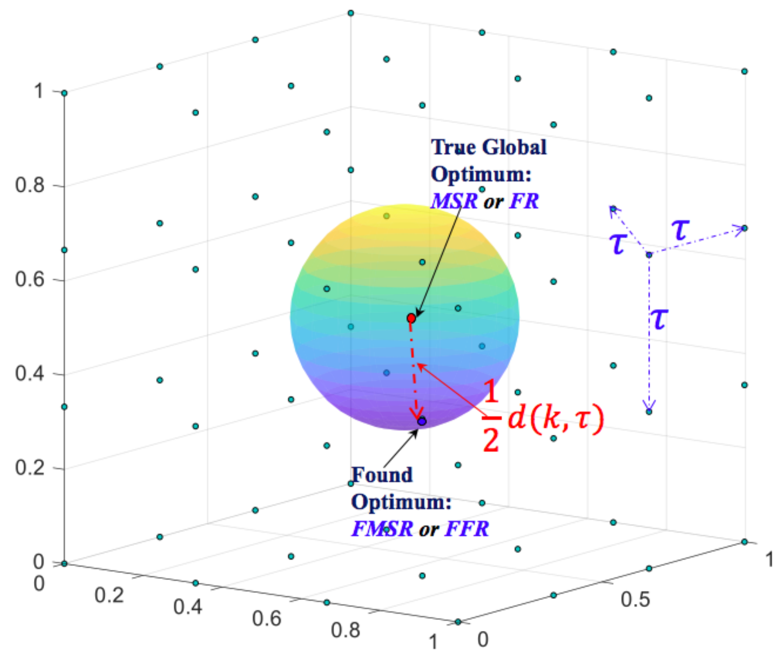


Figure 5.1: Provable guarantees for the MSR and FR_Λ problems on a dense τ-grid (green dots) that is reached upon convergence. In the worst case, the true optimum (the red dot) lies in the middle between two hyper-points of the τ-grid, with the distance of at most $\frac{1}{2}d(L^p, \tau)$ from the found optimum.

Intuitively, if a misclassification aggregator $\boldsymbol{\alpha}'$ with respect to β is classified correctly, then *all* inputs in $\mathsf{Ball}(\boldsymbol{\alpha}', L^p, \beta)$ are classified correctly. And if we specify this β by $\frac{1}{2}d(L^p, \tau)$, then this is exactly what a small subspace covers in Lemma 4. That is, if a grid point is classified correctly, then all other inputs in its metric ball are also classified correctly.

Error Bounds

By utilising the concepts of a τ-grid input and a misclassification aggregator, we can approximate the value of MSR using FMSR. Remember that it is always the maximum safe radius problem that we endeavour to solve. However, because computing MSR directly is NP-hard, we work around this by solving the finite maximum safe radius problem and use FMSR to estimate MSR. Recall from Lemma 3 that we already have MSR $\leq$ FMSR. Below in Lemma 5 (with proof in Appendix B.1) we bound the error by $\frac{1}{2}d(L^p, \tau)$, as illustrated in Figure 5.1.

Lemma 5. *If all τ-grid inputs are misclassification aggregators with respect to $\frac{1}{2}d(L^p,\tau)$, then*

$$\texttt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d) \geq \texttt{FMSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau) - \tfrac{1}{2}d(L^p,\tau). \tag{5.8}$$

Intuitively, this means that the adversarial example $\boldsymbol{\alpha}''$ contributing to the actual `MSR` (i.e., the true optimum), if it exists in the norm ball of the original input $\boldsymbol{\alpha}$, will be covered by the small subspace of a certain τ-grid point, say $\boldsymbol{\alpha}'$ (Lemma 4). And according to Definition 5.7, because $\boldsymbol{\alpha}''$ is misclassified, the fact that $\boldsymbol{\alpha}'$ is a misclassification aggregator will result in itself being misclassified. In other words, the distance from $\boldsymbol{\alpha}'$ to $\boldsymbol{\alpha}$ is in fact the `FMSR` (i.e., the found optimum). Since each τ-grid input has radius $\frac{1}{2}d(L^p,\tau)$, we know that `MSR` is in the same radius of `FMSR`.

In order to make sure that the condition in Lemma 5 is satisfied, i.e., all τ-grid inputs are misclassification aggregators with respect to $\frac{1}{2}d(L^p,\tau)$, we need to determine the manipulation magnitude τ for a Lipschitz continuous network. In the following, we discuss how to compute the largest τ. First, we introduce the concept of a *minimum confidence margin*, which is essentially the discrepancy between the maximum and the next largest confidence of an input dimension for a network.

Definition 5.8 (Minimum Confidence Margin)**.** *Given a network $\mathcal{N}$, an input $\boldsymbol{\alpha}$, and a class c, we define the* minimum confidence margin *between the class c and another class $c' \neq \mathcal{N}(\boldsymbol{\alpha})$ as*

$$\mathsf{Margin}(\boldsymbol{\alpha},c) = \min_{c'\in C, c'\neq c}\{\mathcal{N}(\boldsymbol{\alpha},c) - \mathcal{N}(\boldsymbol{\alpha},c')\}. \tag{5.9}$$

Note that, we can compute $\mathsf{Margin}(\boldsymbol{\alpha},c)$ in constant time. The following lemma shows that the above-mentioned condition about misclassification aggregators can be obtained if τ is sufficiently small.

Lemma 6. *Let $\mathcal{N}$ be a Lipschitz network with a Lipschitz constant Lip_c for every class $c \in C$. If*

$$d(L^p,\tau) \leq \frac{2\cdot\mathsf{Margin}(\boldsymbol{\alpha}',\mathcal{N}(\boldsymbol{\alpha}'))}{\max\limits_{c\in C, c\neq\mathcal{N}(\boldsymbol{\alpha}')}(\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c)} \tag{5.10}$$

for all τ-grid input $\boldsymbol{\alpha}' \in \Gamma(\boldsymbol{\alpha}, L^p, d, \tau)$, then all τ-grid inputs are misclassification aggregators with respect to $\frac{1}{2}d(L^p, \tau)$.

Proof. For any input $\boldsymbol{\alpha}''$ whose closest τ-grid input is $\boldsymbol{\alpha}'$, we have

$$\begin{aligned}
&\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathsf{Margin}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) \\
&= \min_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} \{\mathcal{N}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathcal{N}(\boldsymbol{\alpha}', c)\} - \min_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} \{\mathcal{N}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) - \mathcal{N}(\boldsymbol{\alpha}'', c)\} \\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} \{\mathcal{N}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathcal{N}(\boldsymbol{\alpha}', c) - \mathcal{N}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) + \mathcal{N}(\boldsymbol{\alpha}'', c)\} \\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} \{|\mathcal{N}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathcal{N}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}'))| + |\mathcal{N}(\boldsymbol{\alpha}'', c) - \mathcal{N}(\boldsymbol{\alpha}', c)|\} \\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c) \cdot \|\boldsymbol{\alpha}' - \boldsymbol{\alpha}''\|_p \\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c) \cdot \frac{1}{2} d(L^p, \tau)
\end{aligned} \tag{5.11}$$

Now, to ensure that no class change occurs between $\boldsymbol{\alpha}''$ and $\boldsymbol{\alpha}'$, we need to have $\mathsf{Margin}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) \geq 0$, which means that $\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathsf{Margin}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) \leq \mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))$. Therefore, we can let

$$\max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c) \cdot \frac{1}{2} d(L^p, \tau) \leq \mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')). \tag{5.12}$$

Note that $\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))$ is dependent on the τ-grid input $\boldsymbol{\alpha}'$, and thus can be computed when we construct the grid. Finally, we let

$$d(L^p, \tau) \leq \frac{2 \cdot \mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))}{\max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c)} \tag{5.13}$$

Therefore, if we have the above inequality for every τ-grid input, then we can conclude $\mathsf{Margin}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) \geq 0$ for any $\boldsymbol{\alpha}'' \in \mathsf{Ball}(\boldsymbol{\alpha}', L^p, d)$, i.e., $\mathcal{N}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) \geq \mathcal{N}(\boldsymbol{\alpha}'', c)$ for all $c \in C$. The latter means that no class change occurs. □

Intuitively, only if the manipulation magnitude τ satisfies Inequality (5.10) will the τ-grid points be misclassification aggregators. In other words, when the magnitude is sufficiently small, which contributes to a small enough grid width, all the points in the grid cell will have the same classification as the grid point.

Combining Lemmas 3, 5, and 6, we have the following theorem which shows

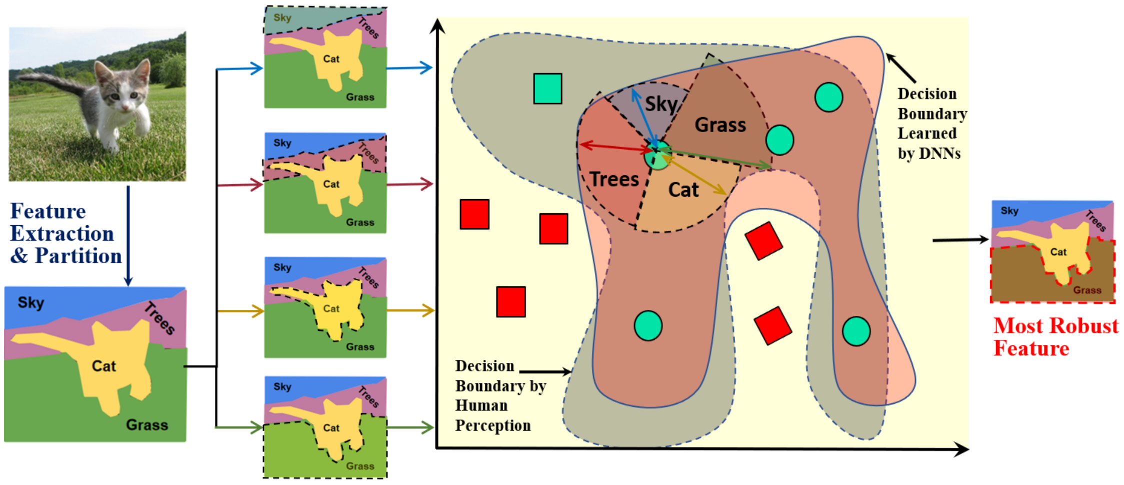

Feature Extraction & Partition
Sky
Trees
Cat
Grass
Decision Boundary Learned by DNNs
Decision Boundary by Human Perception
Most Robust Feature

Figure 5.2: Illustration of the *feature robustness* ($\mathtt{FR}_\Lambda$) problem, which aims to find, on an original image $\boldsymbol{\alpha}$, a feature, or a subset of features, that is the most robust against adversarial perturbations. Given a benign image, we first apply feature extraction or semantic partitioning methods to produce a set of disjoint features ('Sky', 'Trees', 'Cat', etc.), then we find a set of robust features that is most resilient to adversarial perturbations ('Grass' in the figure), which quantifies the most robust direction in a safe norm ball.

that the reduction from the maximum safe radius problem to its finite optimisation has a provable guarantee, depending on the choice of the manipulation magnitude. The proof is technical and given in Appendix B.1.

Theorem 6. *Let $\mathcal{N}$ be a Lipschitz network with a Lipschitz constant* Lip_c *for every class $c \in C$. If*

$$d(L^p, \tau) \leq \frac{2 \cdot \mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))}{\max_{c\in C, c\neq \mathcal{N}(\boldsymbol{\alpha}')}(\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c)} \tag{5.14}$$

for all τ-grid inputs $\boldsymbol{\alpha}' \in \Gamma(\boldsymbol{\alpha}, L^p, d, \tau)$, then we can use $\mathtt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ *to estimate* $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ *with an error bound $\frac{1}{2}d(L^p, \tau)$.*

5.1.2 The Feature Robustness Problem

Partly inspired by the saliency maps of neural networks in Chapter 4, we find it interesting to see how various network models would 'see' the same image differently, even when they all classify it with the same label. Therefore, in order to enable better explainability and interpretability in terms of how networks actually 'understand' an image, we study the *feature robustness* problem, which aims to identify, on an original input, a feature, or a subset of features, that is the most/least robust against adversarial perturbations, illustrated in Figure 5.2. Practically, if we manage to guarantee the safety of the least robust feature against a certain degree of adversarial manipulations, then we can say that for all other features of this input, manipulations within the same degree will definitely not alter the classification of the input. For example, in Figure 5.2 if the 'Cat' feature turns out to be the most vulnerable, and a small ink stain in 'Cat' does not make the image an adversarial example, then the same stain in any other feature will also not change the classification. Apart for this thesis, we mention some other works concerning feature robustness in the domain of explainable AI. For example, [58] explains the decision-making of an image classification network through different contributions of the superpixels (i.e., features) and [43] presents a general additive model for explaining the decisions of a network by the Shapley value computed over the set of features.

Below we introduce the formal definition of the feature robustness problem. In this context, we let $P_0(\boldsymbol{\alpha}, \boldsymbol{\alpha}') \subseteq P_0$ be the set of input dimensions on which $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ have different values.

Definition 5.9 (Feature Robustness)**.** *The* feature robustness *problem is defined as follows.*

$$\mathsf{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) = \max_{\lambda \in \Lambda(\boldsymbol{\alpha})} \{\mathtt{xFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, \lambda, c, L^p, d)\} \tag{5.15}$$

where $\mathtt{xFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}_m, \lambda_m, c, L^p, d) =$

$$\begin{cases} \min\limits_{\boldsymbol{\alpha}_{m+1} \in \mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)} \{\|\boldsymbol{\alpha}_m - \boldsymbol{\alpha}_{m+1}\|_p + \\ \mathsf{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}_{m+1}, c, L^p, d) \mid \emptyset \neq P_0(\boldsymbol{\alpha}_m, \boldsymbol{\alpha}_{m+1}) \subseteq P_{\lambda_m}\}, & \textit{if } \boldsymbol{\alpha}_m \notin \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c) \\ 0, & \textit{otherwise} \end{cases} \tag{5.16}$$

where Λ *is a feature extraction function, and* $\boldsymbol{\alpha}_m, \boldsymbol{\alpha}_{m+1} (m \in \mathbb{N})$ *are the inputs before and after the application of some manipulation on a feature* λ_m*, respectively. If after selecting a feature* λ_m *no adversarial example can be reached, i.e.,* $\forall \boldsymbol{\alpha}_{m+1} : P_0(\boldsymbol{\alpha}_m, \boldsymbol{\alpha}_{m+1}) \subseteq P_{\lambda_m} \Rightarrow \boldsymbol{\alpha}_{m+1} \notin \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$*, then we* *let* $\texttt{xFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}_m, \lambda_m, c, L^p, d) = d + \epsilon.$

Intuitively, the search for the most robust feature alternates between maximising over the features and minimising over the possible input dimensions within the selected feature, with the distance to the adversarial example as the objective. Starting from $\texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}_0, c, L^p, d)$ where $\boldsymbol{\alpha}_0$ is the original image, the process moves to $\texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}_1, c, L^p, d)$ by a max-min alternation on selecting feature λ_0 and next input $\boldsymbol{\alpha}_1$. This continues until either an adversarial example is found, or the next input $\boldsymbol{\alpha}_i$ for some $i > 0$ is outside the d-neighbourhood $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$. The value $d + \epsilon$ is used is to differentiate from the case where the minimal adversarial example has exactly distance d from $\boldsymbol{\alpha}_0$ and the manipulations are within λ_0. In such a case, according to Equation (5.16), we have $\texttt{xFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}_0, \lambda_0, c, L^p, d) = d$.

Assuming $\texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ has been computed and a *distance budget* $d' \leq d$ is given to manipulate the input $\boldsymbol{\alpha}$, the following cases can be considered.

- If $\texttt{FR}_\Lambda > d$, then there are robust features, and if manipulations are restricted to those features no adversarial example is possible.
- If $\texttt{FR}_\Lambda \leq d' \leq d$, then, no matter how one restricts the features to be manipulated, an adversarial example can be found within the budget.
- If $\texttt{MSR} \leq d' < \texttt{FR}_\Lambda \leq d$, then the existence of adversarial examples is controllable, i.e., we can choose a set of features on which the given distance budget d' is insufficient to find an adversarial example. This differs from the first case in that an adversarial example can be found if given a larger d.

Therefore, studying the feature robustness problem enables a better understanding of the robustness of individual features and how the features contribute to the robustness of an image.

It is straightforward to show that

$$\texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) \leq \texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d). \tag{5.17}$$

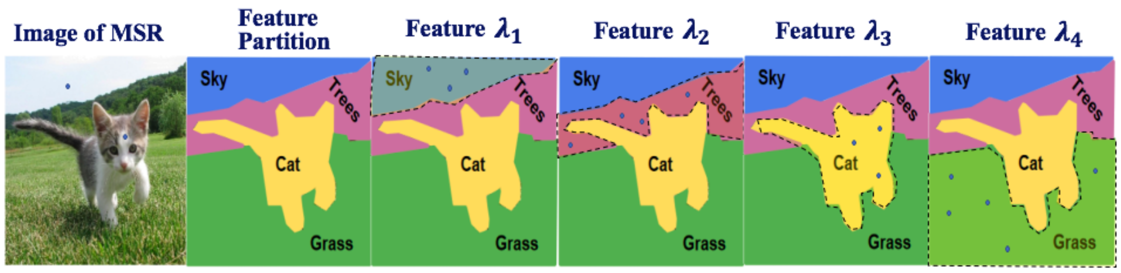


Figure 5.3: Illustration of the *maximum safe radius* (MSR) and *feature robustness* ($\mathtt{FR}_\Lambda$) problems. From left to right: an adversarial example with two pixel changes, feature extraction of the image, adversarial examples with three changed pixels on features 'Sky' and 'Cat', four changed pixels on 'Trees', and five pixel manipulations on 'Grass', respectively.

Compared to the absolute safety radius by $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$, $\mathtt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ can be seen as a *relative* safety radius, within which the existence of adversarial examples can be controlled. Theoretically, the $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ problem can be seen as a special case of the $\mathtt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ problem, when we let $|\Lambda(\boldsymbol{\alpha})| = 1$. We study them separately, because the $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ problem is interesting on its own, and, more importantly, we show later that they can be solved using different methods.

One can also consider a simpler variant of this problem, which aims to find a subset of features that are most resilient to perturbations, and which can be solved by only considering singleton sets of features. We omit the formalisation for reasons of space.

We illustrate the two problems, the *maximum safe radius* (MSR) and (the simpler variant of) *feature robustness* ($\mathtt{FR}'_\Lambda$), through Example 5.

Example 5. *As shown in Figure 5.3, the minimum distance from the original image to an adversary is two pixels, i.e.,* $\mathtt{MSR} = 2$ *(for simplicity here we take the Hamming distance). That is, for a norm ball with radius less than 2, the image is absolutely safe. Note that, for* MSR*, the manipulations can span different features. After feature extraction, we find the* maximum safe radius *of each feature, i.e.,* $\mathtt{MSR}_{\lambda_1} = 3$, $\mathtt{MSR}_{\lambda_2} = 4$, $\mathtt{MSR}_{\lambda_3} = 3$, $\mathtt{MSR}_{\lambda_4} = 5$.

Assume we have a norm ball of radius d, and a distance budget d', then:

- *if $d = 4$, then by definition we have $\mathtt{FR}'_\Lambda = 4 + \epsilon$, i.e., manipulating 'Grass' cannot change the classification;*
- *if $d = 10$ and $d' = 7$ then we have $\mathtt{FR}'_\Lambda = 5 < d' < d$, i.e., all the features are fragile;*
- *if $d = 10$ and $d' = 4$ then $d' < \mathtt{FR}'_\Lambda = 5 < d$, i.e., the existence of an adversary is controllable by restricting perturbations to 'Grass'.*

Approximation Based on Finite Optimisation

Similarly to the case of the maximum safe radius, we reduce the feature robustness problem to finite optimisation by implementing the search for adversarial examples using input manipulations.

Definition 5.10 (Finite Feature Robustness)**.** *Let $\tau \in (0, 1]$ be a manipulation magnitude. The* finite feature robustness *problem $\mathtt{FFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ based on input manipulation is as follows:*

$$\mathtt{FFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) = \max_{\lambda \in \Lambda(\boldsymbol{\alpha})} \{\mathtt{xFFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, \lambda, c, L^p, d, \tau)\} \tag{5.18}$$

where $\mathtt{xFFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}_m, \lambda_m, c, L^p, d, \tau) =$

$$\begin{cases} \min\limits_{X \subseteq P_{\lambda_m}} \min\limits_{\psi \in \Psi} \{\|\boldsymbol{\alpha}_m - \delta_{\tau,X,\psi}(\boldsymbol{\alpha}_m)\|_p + \\ \qquad \mathtt{FFR}_\Lambda(\mathcal{N}, \delta_{\tau,X,\psi}(\boldsymbol{\alpha}_m), c, L^p, d, \tau)\}, & \text{if } \boldsymbol{\alpha}_m \notin \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c) \\ 0, & \text{otherwise} \end{cases} \tag{5.19}$$

where Λ is a feature extraction function, and $\boldsymbol{\alpha}_m, \delta_{\tau,X,\psi}(\boldsymbol{\alpha}_m), m \in \mathbb{N}$, are the perturbed inputs before and after the application of manipulation $\delta_{\tau,X,\psi}$ on a feature λ_m, respectively. If after selecting a feature λ_m no adversarial example can be reached, i.e., $\forall X \subseteq P_{\lambda_m} \forall \psi \in \Psi : \delta_{\tau,X,\psi}(\boldsymbol{\alpha}_m) \notin \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$, then we let $\mathtt{xFFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}_m, \lambda_m, c, L^p, d, \tau) = d + \epsilon$.

Compared to Definition 5.9, the search for another input by $\min_{\boldsymbol{\alpha}_{m+1} \in \mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)}$ is implemented by the combinatorial search over the finite sets of feature sets and instructions.

Error Bounds

The case for the feature robustness problem largely follows that of the maximum safe radius problem. Similarly, we have the following lemma (with proof in Appendix B.1) which bounds the error of $\texttt{FFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ to $\frac{1}{2}d(L^p, \tau)$, which depends on the value of manipulation magnitude τ.

Lemma 7. *If all τ-grid inputs are misclassification aggregators with respect to $\frac{1}{2}d(L^p, \tau)$, then $\texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) \geq \texttt{FFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) - \frac{1}{2}d(L^p, \tau)$.*

Combining Lemmas 3, 6, and 7, we have the following theorem which shows that the reduction has a provable guarantee under the assumption of Lipschitz continuity. The approximation error depends linearly on the prediction confidence on discretised 'grid' inputs and is inversely proportional with respect to the Lipschitz constants of the network. The proof is technical and given in Appendix B.1.

Theorem 7. *Let $\mathcal{N}$ be a Lipschitz network with a Lipschitz constant Lip_c for every class $c \in C$. If*

$$d(L^p, \tau) \leq \frac{2 \cdot \mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))}{\max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')}(\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c)} \tag{5.20}$$

for all τ-grid inputs $\boldsymbol{\alpha}' \in \Gamma(\boldsymbol{\alpha}, L^p, d, \tau)$, then we can use $\texttt{FFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ to estimate $\texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ with an error bound $\frac{1}{2}d(L^p, \tau)$.

5.2 A Game-Based Approximate Verification Approach

In this section, we define a two-player game and show that the solutions of the two finite optimisation problems, $\texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ and $\texttt{FFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$, given in Expressions (5.6) and (5.18) can be reduced to the computation of the rewards of Player $\texttt{I}$ taking an optimal strategy. The two problems differ in that they induce games in which the two players are *cooperative* or *competitive*, respectively.

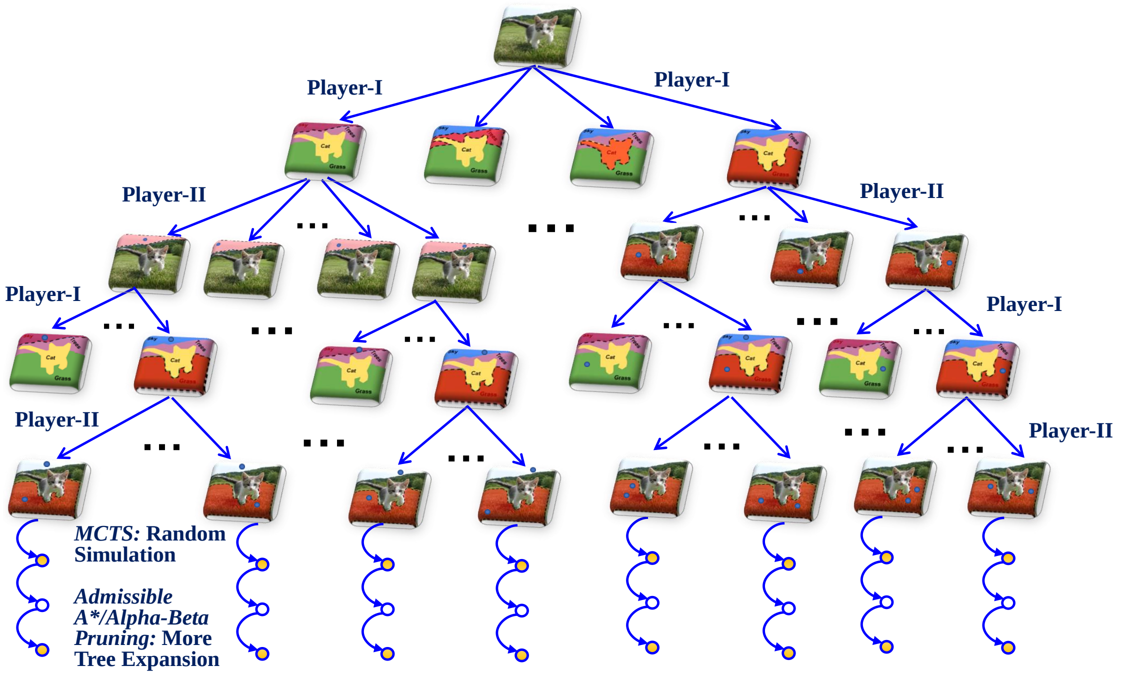


Figure 5.4: The two-player turn-based solution for finite optimisation. Player I selects features and Player II then performs an atomic input manipulation within the selected features. For the *maximum safe radius* problem, both players aim to minimise the distance to an adversarial example; for the *feature robustness* problem, while Player II has the same objective, Player I plays against this, i.e., aiming to prevent the reaching of adversarial examples by taking suitable actions. The game terminates when an adversary is found or the distance budget for adversarial perturbation has been reached.

The game proceeds by constructing a sequence of atomic input manipulations to implement the optimisation objectives in Equations (5.6) and (5.18).

5.2.1 Problem Solving as A Two-Player Turn-Based Game

The game has two players, who take turns to act. Specifically, Player I selects features and Player II then selects an atomic input manipulation within the selected features. While Player II aims to minimise the distance to an adversarial example, depending on the optimisation objective designed for either $\texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ or $\texttt{FFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$, Player I can be *cooperative* or *competitive*. We remark that, in contrast to [76] where the games were originally introduced, we do not consider the nature player. Figure 5.4 illustrates the game model with a partially-expanded game tree.

Definition 5.11 (Game). *Given an input $\boldsymbol{\alpha}$, we let $\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) = (S \cup (S \times \Lambda(\boldsymbol{\alpha})), s_0, \{T_a\}_{a \in \{\mathtt{I}, \mathtt{II}\}}, L)$ be* a game model*, where*

- *S is* a set of game states *belonging to Player* I *such that each state represents an input in* Ball$(\boldsymbol{\alpha}, L^p, d)$*, and $S \times \Lambda(\boldsymbol{\alpha})$ is* a set of game states *belonging to Player* II *where $\Lambda(\boldsymbol{\alpha})$ is a set of features of input $\boldsymbol{\alpha}$. We write $\boldsymbol{\alpha}(s)$ for the input associated to the state $s \in S$.*
- *$s_0 \in S$ is* the initial game state *such that $\boldsymbol{\alpha}(s_0)$ is the original input $\boldsymbol{\alpha}$.*
- Transition relation $T_{\mathtt{I}} : S \times \Lambda(\boldsymbol{\alpha}) \to S \times \Lambda(\boldsymbol{\alpha})$ *is defined as*

$$T_{\mathtt{I}}(s, \lambda) = (s, \lambda), \tag{5.21}$$

and transition relation $T_{\mathtt{II}} : (S \times \Lambda(\boldsymbol{\alpha})) \times \mathcal{P}(P_0) \times \Psi \to S$ *is defined as*

$$T_{\mathtt{II}}((s, \lambda), X, \psi) = \delta_{\tau, X, \psi}(\boldsymbol{\alpha}(s)), \tag{5.22}$$

where $X \subseteq P_\lambda$ is a set of input dimensions within feature λ, $\psi : P_0 \to \{-1, +1\}$ is a manipulation instruction, and $\delta_{\tau,X,\psi}$ is an atomic dimension manipulation as defined in Definition 5.2. Intuitively, in every game state $s \in S$, Player I *will choose a feature λ, and, in response to this, Player* II *will choose an atomic input manipulation $\delta_{\tau,X,\psi}$.*
- *The* labelling function $L : S \cup (S \times \Lambda(\boldsymbol{\alpha})) \to C$ *assigns to each state s or (s, λ) a class $\mathcal{N}(\boldsymbol{\alpha}(s))$.*

Intuitively, the game model starts from an input as the initial game state, from which Player I selects features according to its transition relation in Equation (5.21) and Player II determines an atomic manipulation within the chosen features, as reflected by its transition relation in Equation (5.22). When the game proceeds as the two players take turns to act, the game tree gradually expands, as illustrated in Figure 5.4. Note that each Player I's state represents a (perturbed) input in the metric ball of the original input, whereas Player II's states denote feature selection. And the labelling function simply classifies the input points.

Strategy Profile

A path (or game play) of the game model is a sequence $s_1 u_1 s_2 u_2 \ldots$ of game states such that, for all $k \geq 1$, we have $u_k = T_{\texttt{I}}(s_k, \lambda_k)$ for some feature λ_k and $s_{k+1} = T_{\texttt{II}}((s_k, \lambda_k), X_k, \psi_k)$ for some (X_k, ψ_k). Let $last(\rho)$ be the last state of a finite path ρ, and $Path_a^F$ be the set of finite paths such that $last(\rho)$ belongs to player $a \in \{\texttt{I}, \texttt{II}\}$.

Definition 5.12 (Strategy)**.** *A stochastic* strategy $\sigma_{\texttt{I}} : Path_{\texttt{I}}^F \to \mathcal{D}(\Lambda(\boldsymbol{\alpha}))$ *of Player* I *maps each finite path to a distribution over the next actions, and similarly for* $\sigma_{\texttt{II}} : Path_{\texttt{II}}^F \to \mathcal{D}(\mathcal{P}(P_0) \times \Psi)$ *for Player* II*. We call* $\sigma = (\sigma_{\texttt{I}}, \sigma_{\texttt{II}})$ *a* strategy profile*.*

Intuitively, this means that, in every Player I's state, there is a distribution over the features to select, and similarly, when Player II takes its turn, atomic dimension manipulations are imposed according to a distribution. Besides, we also mention that a strategy σ is deterministic if $\sigma(\rho)$ is a Dirac distribution, and is memoryless if $\sigma(\rho) = \sigma(last(\rho))$ for all finite paths ρ.

Rewards

We define a reward $R(\sigma, \rho)$ for a given strategy profile $\sigma = (\sigma_{\texttt{I}}, \sigma_{\texttt{II}})$ and a finite path $\rho \in \bigcup_{a \in \{\texttt{I},\texttt{II}\}} Path_a^F$. The idea of the reward is to accumulate the distance to the adversarial example found over a path. Note that, given σ, the game becomes a deterministic system. Let $\boldsymbol{\alpha}'_\rho = \boldsymbol{\alpha}(last(\rho))$ be the input associated with the last state of the path ρ. We write

$$tc(\rho) \equiv (\mathcal{N}(\boldsymbol{\alpha}'_\rho) = c) \vee (\left\|\boldsymbol{\alpha}'_\rho - \boldsymbol{\alpha}\right\|_p > d), \tag{5.23}$$

representing that the path has reached a state whose associated input either is in the target class c or lies outside the region $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$. The path ρ can be terminated whenever $tc(\rho)$ is satisfied. It is not difficult to see that, due to the constraints in Definition 5.1, every infinite path has a finite prefix which can be terminated (that is, either when an adversarial example is found or the distance to the original image has exceeded d). During each expansion of the game model, an atomic manipulation is employed, which excludes the possibility that an input dimension is perturbed in smaller and smaller steps.

Definition 5.13 (Reward). *Given a strategy profile $\sigma = (\sigma_{\texttt{I}}, \sigma_{\texttt{II}})$ and a finite path ρ, we define a* reward *function as follows:* $R(\sigma, \rho) =$

$$\begin{cases} \left\| \boldsymbol{\alpha}'_\rho - \boldsymbol{\alpha} \right\|_p, & \text{if } tc(\rho), \rho \in Path^F_{\texttt{I}} \\ \sum\limits_{\lambda \in \Lambda(\boldsymbol{\alpha})} \sigma_{\texttt{I}}(\rho)(\lambda) \cdot R(\sigma, \rho T_{\texttt{I}}(last(\rho), \lambda)), & \text{if } \neg tc(\rho), \rho \in Path^F_{\texttt{I}} \\ \sum\limits_{(X,\psi) \in \mathcal{P}(P_0) \times \Psi} \sigma_{\texttt{II}}(\rho)(X, \psi) \cdot R(\sigma, \rho T_{\texttt{II}}(last(\rho), X, \psi)), & \text{if } \rho \in Path^F_{\texttt{II}} \end{cases} \tag{5.24}$$

where $\sigma_{\texttt{I}}(\rho)(\lambda)$ is the probability of selecting feature λ on finite path ρ by Player I*, and $\sigma_{\texttt{II}}(\rho)(X, \psi)$ is the probability of selecting atomic input manipulation $\delta_{\tau,X,\psi}$ based on ρ by Player* II*. The expression $\rho T_{\texttt{I}}(last(\rho), \lambda)$ is the resulting path of Player* I *selecting λ, and $\rho T_{\texttt{II}}(last(\rho), X, \psi)$ is the resulting path of Player* II *applying $\delta_{\tau,X,\psi}$ on $\boldsymbol{\alpha}'_\rho$. We note that a path only terminates on Player* I *states.*

Intuitively, if an adversarial example is found then the reward assigned is the distance to the original input, otherwise it is the weighted summation of the rewards of its children.

Players' Objectives

Players' strategies are to maximise their rewards in a game. The following players' objectives are designed to match the finite optimisation problems stated in Equations (5.6) and (5.18).

Definition 5.14 (Players' Objectives). *In a game, Player* II *chooses a strategy $\sigma_{\texttt{II}}$ to minimise the reward $R((\sigma_{\texttt{I}}, \sigma_{\texttt{II}}), s_0)$, whilst Player* I *has different goals depending on the optimisation problem under consideration.*

- *For the* maximum safe radius *problem, Player* I *chooses a strategy $\sigma_{\texttt{I}}$ to minimise the reward $R((\sigma_{\texttt{I}}, \sigma_{\texttt{II}}), s_0)$, based on the strategy $\sigma_{\texttt{II}}$ of Player* II*. That is, the two players are* cooperative*.*

- *For the* feature robustness *problem, Player* I *chooses a strategy* $\sigma_{\mathtt{I}}$ *to maximise* $R((\sigma_{\mathtt{I}}, \sigma_{\mathtt{II}}), s_0)$*, based on the strategy* $\sigma_{\mathtt{II}}$ *of Player* II*. That is, the two players are* competitive*.*

The goal of the game is for Player I to choose a strategy $\sigma_{\mathtt{I}}$ to optimise its objective, to be formalised below.

5.2.2 Safety Guarantees via Optimal Strategy

For different objectives $x \in \{\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \mathtt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)\}$ of Player I, we construct different games. Given a game model $\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ and an objective x of Player I, there exists an *optimal strategy profile* $\sigma = (\sigma_{\mathtt{I}}, \sigma_{\mathtt{II}})$, obtained by both players optimising their objectives. We will consider the algorithms to compute the optimal strategy profile in Section 5.3. Here we focus on whether the obtained optimal strategy profile σ is able to implement the finite optimisation problems in Equations (5.6) and (5.18).

First of all, we formally define the goal of the game.

Definition 5.15 (Game Goal)**.** *Given a game model* $\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$*, an objective* x *of Player* I*, and an optimal strategy profile* $\sigma = (\sigma_{\mathtt{I}}, \sigma_{\mathtt{II}})$*,* the goal of the game *is to compute the value*

$$val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), x) = R(\sigma, s_0) \tag{5.25}$$

That is, the goal is to compute the reward of the initial state s_0 *based on* σ*. Note that an initial state* s_0 *is also a finite path, and it is a Player* I *state.*

We have the following Theorems 8 and 9 to confirm that the game can return the optimal values for the two finite optimisation problems. Detailed proofs for both theorems are in Appendix B.1.

Theorem 8. *Assume that Player* I *has the objective* $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$*. Then*

$$val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)) = \mathtt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) \tag{5.26}$$

Theorem 9. *Assume that Player* $\mathtt{I}$ *has the objective* $\mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)$. *Then*

$$val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)) = \mathtt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau) \tag{5.27}$$

Combining Theorems 8, 9 with Theorems 6, 7, we have the following corollary, which states that the optimal game strategy is able to achieve the optimal value for the maximum safe radius problem $\mathtt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)$ and the feature robustness problem $\mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)$ with an error bound $\frac{1}{2}d(L^p,\tau)$.

Corollary 1. *The two-player turn-based game is able to solve the* maximum safe radius *problem of Equation (5.5) and the* feature robustness *problem of Equation (5.15) with an error bound* $\frac{1}{2}d(L^p,\tau)$, *when the manipulation magnitude* τ *is such that*

$$d(L^p,\tau) \leq \frac{2\cdot \mathsf{Margin}(\boldsymbol{\alpha}',\mathcal{N}(\boldsymbol{\alpha}'))}{\max_{c'\in C, c'\neq\mathcal{N}(\boldsymbol{\alpha}')}(\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')}+\mathsf{Lip}_{c'})} \tag{5.28}$$

for all τ*-grid inputs* $\boldsymbol{\alpha}'\in\Gamma(\boldsymbol{\alpha},L^p,d,\tau)$.

Furthermore, we have the following lemma.

Lemma 8. *For a game model* $\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)$ *with goal* $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),$ $\mathtt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d))$, deterministic and memoryless strategies *suffice for Player* $\mathtt{I}$*; and similarly for game* $\mathcal{G}$ *with goal* $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d))$.

5.2.3 Complexity of the Problem

As a by-product of Lemma 8, the theoretical complexity of the problems is in PTIME, with respect to the size of the game model $\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)$. However, the size of the game is exponential with respect to the number of input dimensions. More specifically, we have the following complexity result with respect to the manipulation magnitude τ, the pre-specified range size d, and the number of input dimensions m.

Theorem 10 (Complexity)**.** *Given a game model* $\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)$, *the computational time for the two goal values* $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),x)$, *where* $x\in\{\mathtt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\ \mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)\}$, *is* polynomial *with respect to* d/τ

and exponential *with respect to m.*

Proof. We can see that the size of the grid, measured as the number $|\Gamma(\boldsymbol{\alpha}, L^p, d, \tau)|$ of τ-grid inputs in $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$, is polynomial with respect to d/τ and exponential with respect to m. From a τ-grid to any of its neighbouring τ-grids, each player needs to take a move. Therefore, the number of game states is doubled (i.e., polynomial) over $|\Gamma(\boldsymbol{\alpha}, L^p, d, \tau)|$. This yields PTIME complexity of solving the game. □

Considering that the problem instances we work with usually have a large input dimensionality, this complexity suggests that directly working with the explicit game models is impractical. If we consider an alternative representation of a game tree (i.e., an unfolded game model) of finite depth to express the complexity, the number of nodes on the tree is $O(m^h)$ for h the length of the longest finite path without a terminating state. While the precise size of $O(m^h)$ is dependent on the problem (including the image $\boldsymbol{\alpha}$ and the difficulty of crafting an adversarial example), it is roughly $O(50000^{100})$ for the images used in the ImageNet competition and $O(1000^{20})$ for smaller images such as GTSRB, CIFAR-10, and MNIST. This is beyond the capability of existing approaches for exact or ϵ-approximate computation of probability (e.g., reduction to linear programming [67], value iteration, and policy iteration, etc.) that are used in probabilistic verification.

5.3 Algorithms and Implementation

In this section we describe the implementation of the game-based approach introduced in this chapter. Figure 6.3 presents an overview of the reductions from the original problems to the solution of a two-player game for the case of Lipschitz networks, described in Section 5.1. Because exact computation of optimal rewards is computationally hard (Theorem 10), we approximate the rewards by means of algorithms that unfold the game tree based on Monte Carlo tree search (MCTS), Admissible A*, and Alpha-Beta Pruning.

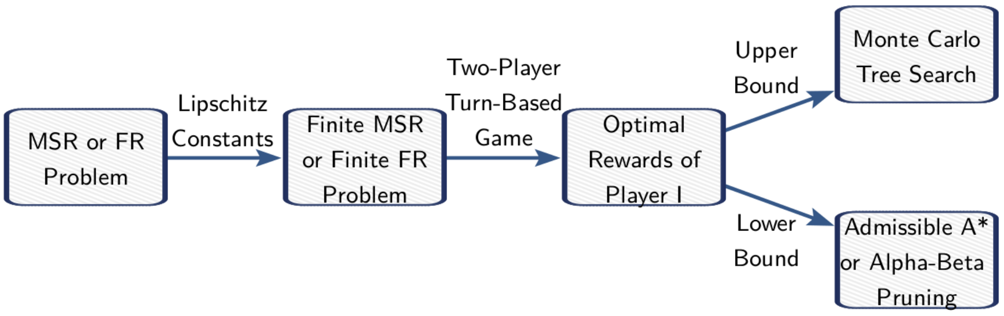

MSR or FR Problem
Lipschitz Constants
Finite MSR or Finite FR Problem
Two-Player Turn-Based Game
Optimal Rewards of Player I
Upper Bound
Monte Carlo Tree Search
Lower Bound
Admissible A* or Alpha-Beta Pruning

Figure 5.5: A game-based approximate verification approach for the *maximum safe radius* (`MSR`) and *feature robustness* (`FR`) problems of deep neural networks.

We take a principled approach to compute for each of the two game values, $\mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ and $\mathtt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$, an upper bound and a lower bound. Our algorithms can gradually, but strictly, improve the bounds, so that they gradually converge as the computation proceeds. For $x \in \{\mathtt{MSR}, \mathtt{FR}_\Lambda\}$, we write l_x and u_x for their lower and upper bound, respectively. The bounds can be interesting in their own. For example, a lower bound $\mathsf{l}_{\mathtt{MSR}}$ suggests absolute safety of an L^p norm ball with radius $\mathsf{l}_{\mathtt{MSR}}$ from the original input $\boldsymbol{\alpha}$, and an upper bound $\mathsf{u}_{\mathtt{MSR}}$ suggests the existence of an adversarial example $\boldsymbol{\alpha}'$ such that $\|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_p = \mathsf{u}_{\mathtt{MSR}}$. On the other hand, given distance budget d', $\mathsf{u}_{\mathtt{FR}_\Lambda} \leq d'$ indicates an unsafe distance from which the existence of adversarial examples is not controllable.

Next we present the algorithms we employ to compute the upper and lower bounds of the values of the games, as well as their convergence analysis.

5.3.1 Upper Bounds: Monte Carlo Tree Search

We present an approach based on Monte Carlo tree search (MCTS) [10] to find an optimal strategy asymptotically. As a heuristic search algorithm for decision processes notably employed in game play, MCTS focuses on analysing the most promising moves via expanding the search tree based on random sampling of the search space. The algorithm, whose pseudo-code is presented in Algorithm 1, gradually expands a *partial game tree* by sampling the strategy space of the model

Algorithm 1: Monte Carlo Tree Search for DNN Verification

```
    Input  : A game model $\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$, a termination condition $tc$
    Output : $val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \mathtt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$ or
             $val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \mathtt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$
1   procedure MCTS($\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), tc$):
2       $root \leftarrow s_0$ ;
3       while ($\neg tc$) do
4           $leaf \leftarrow selection(root)$ ;
5           $newnodes \leftarrow expansion(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), leaf)$ ;
6           for $node$ in $newnodes$ do
7               $v \leftarrow Simulation(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), node)$ ;
8               $backPropogation(node, v)$ ;
9       return optimal value of the $root$ node
```

$\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$. With the upper confidence bound (UCB) [32] as the exploration-exploitation trade-off, MCTS has a theoretical guarantee that it converges to the optimal solution when the game tree is fully explored. In the following, we explain the components of the algorithm.

Concerning the data structure, we maintain the set of nodes on the *partial tree* $\mathcal{T}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$. For every node o on the partial tree, we maintain three variables, r_o, n_o, e_o, which represent the accumulated reward, the number of visits, and the current best input with respect to the objective of the player, respectively. We remark that e_o is usually different from $\boldsymbol{\alpha}(s_o)$, which is the input associated with the game state s_o. Moreover, for every node o, we record its parent node p_o and a set $\mathcal{C}_o$ of its children nodes. The value $val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), x)$ of the game is approximated by $\|e_{root} - \boldsymbol{\alpha}\|_p$, which represents the distance between the original input and the current best input maintained by the root node of the tree.

The *selection* procedure starts from the *root* node, which contains the original image, and conducts a tree traversal until reaching a $leaf$ node (Line 6). From a node, the next child node to be selected is dependent on an exploration-exploitation balance, i.e., UCB [32]. More specifically, on a node o, for every child node

$o' \in \mathcal{C}_o$, we let

$$v(o, o') = \frac{d * n_{o'}}{r_{o'}} + \sqrt{\frac{2 \ln n_o}{n_{o'}}} \tag{5.29}$$

be the weight of choosing o' as the next node from o. Then the actual choice of a next node is conducted by sampling over a probabilistic distribution $Prob_o : \mathcal{C}_o \to [0, 1]$ such that

$$Prob_o(o') = \frac{v_{o,o'}}{\sum_{o' \in \mathcal{C}_o} v_{o,o'}} \tag{5.30}$$

which is a normalisation over the weights of all children. On a *leaf* node o, the *expansion* procedure returns a set of children nodes $\mathcal{C}_o$ by applying the transition relation in the game model $\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ (Line 7). These new nodes are added into the partial tree $\mathcal{T}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$. This is the only way for the partial tree to grow. After expanding the leaf node to have its children added to the partial tree, we call the *Simulation* procedure on every child node (Line 9). A simulation on a new node o is a play of the game from o until it terminates. Players act randomly during the simulation. Every simulation terminates when reaching a terminal node $\boldsymbol{\alpha}'$. Once a terminal node $\boldsymbol{\alpha}'$ is reached, a reward $\|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_p$ can be computed. This reward, together with the input $\boldsymbol{\alpha}'$, is then *backpropagated* from the new child node through its ancestors until reaching the root (Line 10). Every time a new reward v is backpropagated through a node o, we update its associated reward r_o into $r_o + v$ and increase its number of visits into $n_o + 1$. The update of current best input e_o depends on the player who owns the node. For the $\texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ game, e_o is made equivalent to $e_{o'}$ such that

$$o' = \arg\min_{o_1 \in \mathcal{C}_o} \|\boldsymbol{\alpha} - e_{o_1}\|_p \tag{5.31}$$

For the $\texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$ game, Player $\texttt{II}$ also takes the above approach, i.e., Equation (5.31), to update e_o, but for Player $\texttt{I}$ we let e_o be $e_{o''}$ such that

$$o'' = \arg\max_{o_1 \in \mathcal{C}_o} \|\boldsymbol{\alpha} - e_{o_1}\|_p \tag{5.32}$$

We remark the game is not zero-sum for the maximum safe radius problem.

5.3.2 Lower Bounds: Admissible A* in a Cooperative Game

To produce the lower bounds with guarantees, we consider algorithms which can compute optimal strategy deterministically, without relying on the asymptotic convergence as MCTS does. Recall that, to address the maximum safe radius and the feature robustness problems, we have set the game to be *cooperative* and *competitive*, respectively. As Player `I` has opposite goals depending on the game type – to *minimise* the reward in a cooperative game and to *maximise* it when the game is competitive, we utilise different algorithms to compute the lower bounds in each game type. Specifically, in this section we exploit Admissible A* to achieve the lower bound of Player `I` reward when it is *cooperative*, i.e., $\texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$, and in Section 5.3.3 we use Alpha-Beta Pruning to obtain the lower bound of Player `I` reward when it is *competitive*, i.e., $\texttt{FR}_{\Lambda}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$.

The A* algorithm gradually unfolds the game model into a tree. It maintains a set of leaf nodes of the unfolded partial tree, computes an estimate for every node in the set, and selects the node with the least estimated value to expand. The estimation consists of two components, one for the exact cost up to now and the other for the estimated cost of reaching the goal node. In our case, for each game state s, we assign an estimated distance value

$$\mathit{distances}(s) = \left\lVert \boldsymbol{\alpha}(s) - \boldsymbol{\alpha}(s_0) \right\rVert_p + \mathit{heuristic}(\boldsymbol{\alpha}(s)) \tag{5.33}$$

where the first component $\left\lVert \boldsymbol{\alpha}(s) - \boldsymbol{\alpha}(s_0) \right\rVert_p$ represents the distance from the initial state s_0 to the current state s, and the second component $\mathit{heuristic}(\boldsymbol{\alpha}(s))$ denotes the estimated distance from the current state s to a terminal state.

An *admissible* heuristic function is to, given a current input, never overestimate the cost of reaching the terminal game state. Therefore, to achieve the lower bound, we need to take an admissible heuristic function. We remark that, if the heuristic function is inadmissible (i.e., does not guarantee the underestimation of the cost), then the A* algorithm cannot be used to compute the lower bound, but instead can be used to compute the upper bound.

We utilise the minimum confidence margin $\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))$ defined in Definition 5.8 to obtain an admissible heuristic function.

Lemma 9. *For any game state s such that $\boldsymbol{\alpha}(s) = \boldsymbol{\alpha}'$, the following heuristic function is admissible:*

$$heuristic(\boldsymbol{\alpha}') = \frac{\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))}{\max\limits_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c)} \tag{5.34}$$

Proof. Consider the expression $\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathsf{Margin}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}'))$, where $\boldsymbol{\alpha}'$ is the current state and $\boldsymbol{\alpha}''$ is the last state before a terminal state. Then we have that

$$\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathsf{Margin}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) \leq \mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) \tag{5.35}$$

Now because

$$\begin{aligned}
&\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathsf{Margin}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) \\
&= \min_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} \{\mathcal{N}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathcal{N}(\boldsymbol{\alpha}', c)\} - \min_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} \{\mathcal{N}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) - \mathcal{N}(\boldsymbol{\alpha}'', c)\} \\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} \{|\mathcal{N}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) - \mathcal{N}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}'))| + |\mathcal{N}(\boldsymbol{\alpha}'', c) - \mathcal{N}(\boldsymbol{\alpha}', c)|\} \\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c) \cdot \|\boldsymbol{\alpha}' - \boldsymbol{\alpha}''\|_p
\end{aligned} \tag{5.36}$$

we can let

$$\max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_j) \cdot \|\boldsymbol{\alpha}' - \boldsymbol{\alpha}''\|_p \leq \mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}')) \tag{5.37}$$

Thus, we define

$$heuristic(\boldsymbol{\alpha}') = \frac{\mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))}{\max\limits_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c)} \tag{5.38}$$

which is sufficient to ensure that $\mathsf{Margin}(\boldsymbol{\alpha}'', \mathcal{N}(\boldsymbol{\alpha}')) \geq 0$ for any $\boldsymbol{\alpha}''$. That is, the distance $heuristic(\boldsymbol{\alpha}')$ is a lower bound of reaching a misclassification. □

The Admissible A* algorithm is presented in Algorithm 2. In the following, we explain the main components of the algorithm. For each *root* node (initialised as the original input), Player `I` chooses between mutually exclusive *features* partitioned based on either the grey-box or black-box approach. Subsequently, in each *feature*,

Algorithm 2: Admissible A* for DNN Verification

```
   Input  : A game model $\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$, a termination condition $tc$
   Output : $val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)), \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$
1  procedure AdmissibleA*($\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), tc$):
2      $root \leftarrow s_0$ ;
3      while $(\neg tc)$ do
4          $features \leftarrow$ Player I ($root$, feature extraction = grey/black) ;
5          for $feature$ in $features$ do
6              $dimensions \leftarrow$ Player II ($feature$) ;
7              $newnodes \leftarrow AtomicManipulation(dimensions)$ ;
8              for $node$ in $newnodes$ do
9                  $distances \leftarrow DistanceEstimation(node)$ ;
10         $root \leftarrow \texttt{MSR}(distances)$ ;
11     return $\|\boldsymbol{\alpha}(root) - \boldsymbol{\alpha}(s_0)\|_p$
```

Player II chooses among all the *dimensions* within each feature (Line 4-8). On each of the *dimensions*, an *AtomicManipulation* is constructed and applied. We add $+\tau$ and $-\tau$ to each dimension, and make sure that it does not exceed the upper and lower bounds of the input dimension, e.g., 1 and 0 if the input is pre-processed (normalised). If exceeded, the bound value is used instead. This procedure essentially places adversarial perturbations on the image, and all manipulated images become the *newnodes* (Line 9). For each *node* in the *newnodes*, the *DistanceEstimation* function in Equation (5.33) is used to compute a value, which is then added into the set *distances*. The set *distances* maintains the estimated values for all leaf nodes (Line 10-11). Among all the leaf nodes whose values are maintained in *distances*, we select the one with the minimum MSR as the new *root* (Line 12).

As for the termination condition $\neg tc$, the algorithm gradually unfolds the game tree with increasing tree depth $td = 1, 2, \dots$ and go on. Because all nodes on the same level of the tree have the same distance to the original input $\boldsymbol{\alpha}$, every tree depth $td > 0$ is associated with a distance $d(td)$, such that $d(td)$ is the distance of the nodes at level td. For a given tree depth td, we have a termination condition $tc(td)$ requiring that either

- all the tree nodes up to depth td have been explored, or

- the current *root* is an adversarial example.

For the latter condition, $\|\boldsymbol{\alpha}(root) - \boldsymbol{\alpha}(s_0)\|_p$ is returned and the algorithm converges. As for the former, we update $d(td)$ as the current lower bound of the game value $val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$. Note that the termination condition guarantees the closest adversarial example that corresponds to FMSR, which is within distance $\frac{1}{2}d(L^p, \tau)$ from the actual closest adversarial example corresponding to MSR.

5.3.3 Lower Bounds: Alpha-Beta Pruning in a Competitive Game

Alpha-Beta Pruning is an adversarial search algorithm, applied commonly in two-player games, to minimise the possible cost in a maximum cost scenario. In this chapter, we apply Alpha-Beta Pruning to compute the lower bounds of Player I reward in a *competitive* game, i.e., $\texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$.

Lemma 10. *For any game state $s \in S \cup (S \times \Lambda(\boldsymbol{\alpha}))$, we let $\sigma_{\texttt{I}}(s) \in S \times \Lambda(\boldsymbol{\alpha})$ be the next state of $s \in S$ after Player I taking an action $\sigma_{\texttt{I}}$, and $\sigma_{\texttt{II}}(s) \in S$ be the next state of $s \in S \times \Lambda(\boldsymbol{\alpha})$ after Player II taking an action $\sigma_{\texttt{II}}$. If using alpha(s) (initialised as $-\infty$) to denote Player I current maximum reward on state s and beta(s) (initialised as $+\infty$) to denote Player II current minimum reward on state s, and let*

$$\texttt{alpha}(s) = \max_{\sigma_{\texttt{I}}} \texttt{beta}(\sigma_{\texttt{I}}(s)) \qquad \text{if } s \in S \tag{5.39}$$

$$\texttt{beta}(s) = \min_{\sigma_{\texttt{II}}} \texttt{alpha}(\sigma_{\texttt{II}}(s)) \qquad \text{if } s \in S \times \Lambda(\boldsymbol{\alpha}) \tag{5.40}$$

then alpha(s_0) is a lower bound of $val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$.

Note that, for a game state s, whenever $\texttt{alpha}(s) \geq \texttt{beta}(s')$ for some $s' = \sigma_{\texttt{I}}(s)$ is satisfied, Player I does not need to consider the remaining strategies of Player II on state s', as such will not affect the final result. This is the pruning of the game tree. The Alpha-Beta Pruning algorithm is presented in Algorithm 3. Many components of the algorithm are similar to those of Admissible A*, except that each node maintains two values: alpha value and beta value. For every node, its

Algorithm 3: Alpha-Beta Pruning for DNN Verification

```
Input  : A game model $\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$, a termination condition $tc$
Output : $val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)), \texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$
procedure AlphaBeta($\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), tc$):
    $root \leftarrow s_0$ ;
    $root$.alpha $\leftarrow -\infty$ ;
    $features \leftarrow$ Player I ($root$, feature extraction = grey/black) ;
    for $feature$ in $features$ do
        $feature$.beta $\leftarrow +\infty$ ;
        $dimensions \leftarrow$ Player II ($feature$) ;
        $newnodes \leftarrow AtomicManipulation(dimensions)$ ;
        for $node$ in $newnodes$ do
            if $tc$ then return $\|\boldsymbol{\alpha}(node) - \boldsymbol{\alpha}(s_0)\|_p$;
            else $node$.alpha $\leftarrow$ AlphaBeta($node, tc$);
        $feature$.beta $\leftarrow \min(newnodes.$alpha$)$ ;
    $root$.alpha $\leftarrow \max(features.$beta$)$ ;
    return $root$.alpha
```

alpha value is initialised as $-\infty$ and its beta value is initialised as $+\infty$. For each *feature*, its beta value is the minimum of all the alpha values of the perturbed inputs whose manipulated dimensions are within this feature (Line 14); for *root* in each recursion, the alpha value is the maximum of all the beta values of the features (Line 15). Intuitively, beta maintains the MSR of each feature, while alpha maintains the $\texttt{FR}_\Lambda$ of an input.

5.3.4 Anytime Convergence

In this section, we show the convergence of our approach, i.e., that both bounds are monotonically improved with respect to the optimal values.

Upper Bounds: $1/\epsilon$-Convergence and Practical Termination Condition

Because we are working with a finite game, MCTS is guaranteed to converge when the game tree is fully expanded, but the worst case convergence time may be prohibitive. In practice, we can work with $1/\epsilon$-convergence by letting the program terminate when the current best bound has not been improved for e.g., $\lceil 1/\epsilon \rceil$ iterations, where

$\epsilon > 0$ is a small real number. We can also impose time constraint tc, and ask the program to return once the elapsed time of the computation has exceeded tc.

In the following, we show that the intermediate results from Algorithm 1 can be the upper bounds of the optimal values, and the algorithm is continuously improving the upper bounds, until the optimal values are reached.

Lemma 11. *Let $\|\boldsymbol{\alpha}' - e_{root}\|_p$ be the returned result from Algorithm 1. For an $\texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ game, we have that*

$$\|\boldsymbol{\alpha}' - e_{root}\|_p \geq val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)). \tag{5.41}$$

Moreover, we reamrk that the discrepancy between $\|\boldsymbol{\alpha}' - e_{root}\|_p$ and the goal value $val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$ improves monotonically as the computation proceeds.

Proof. Assume that we have a partial tree $\mathcal{T}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$. We prove by induction on the structure of the tree. As the base case, for each leaf node o we have that its best input e_o is such that

$$\|\boldsymbol{\alpha} - e_o\|_p \geq val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)) \tag{5.42}$$

because a random simulation can always return a current best, which is an upper bound to the global optimal value. The equivalence holds when the simulation found an adversarial example with minimum distance.

Now, for every internal node o, by Equation (5.31) we have that

$$\exists o_1 \in \mathcal{C}_o : \|\boldsymbol{\alpha} - e_o\|_p \geq \|\boldsymbol{\alpha} - e_{o_1}\|_p \tag{5.43}$$

which, together with Equation (5.42) and induction hypothesis, implies that $\|\boldsymbol{\alpha} - e_o\|_p \geq val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$. Equation (5.41) holds since the root node is also an internal node.

The monotonic improvement can be seen from Equation (5.31), namely that, when, and only when, the discrepancy for the leaf node is improved after a new round of random simulation, can the discrepancy for the root node be improved. Otherwise, it remains the same. □

Similarly, we have the following lemma for the feature robustness game.

Lemma 12. *Let $\|\boldsymbol{\alpha}' - e_{root}\|_p$ be the returned result from Algorithm 1. For an $\mathtt{FFR}_\Lambda(L^p, d, \boldsymbol{\alpha}, c)$ game, we have that*

$$\|\boldsymbol{\alpha}' - e_{root}\|_p \geq val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \mathtt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)). \tag{5.44}$$

Proof. The proof is similar to that of Lemma 11, except that, according to Equation (5.32), for the nodes of Player $\mathtt{I}$ (including the root node) to reduce the discrepancy, i.e., $\|\boldsymbol{\alpha} - e_o\|_p - val(\mathcal{G}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d), \mathtt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d))$, it requires that all its children nodes reduce their discrepancy. □

Lower Bounds: Gradual Expansion of the Game Tree

The monotonicity of the lower bounds is achieved by gradually increasing the tree depth *td*. Because, in both algorithms, the termination conditions are the full exploration of the partial trees up to the depth *td*, it is straightforward that the results returned by the algorithms are either the lower bounds or the converged results.

5.4 Experimental Results

This section presents the experimental results for the proposed game-based framework, implemented as a tool DeepGame, for the safety verification of deep neural networks, focused on demonstrating the convergence of the upper and lower bounds, and the comparison with the state-of-the art techniques. The hardware environment is a Linux server with NVIDIA GeForce GTX TITAN Black GPUs, and the operating system is Ubuntu 14.04.3 LTS.

5.4.1 Feature-Based Partitioning

Our game-based approach, where Player $\mathtt{I}$ determines features and Player $\mathtt{II}$ selects pixels or dimensions within the selected feature, requires an appropriate feature

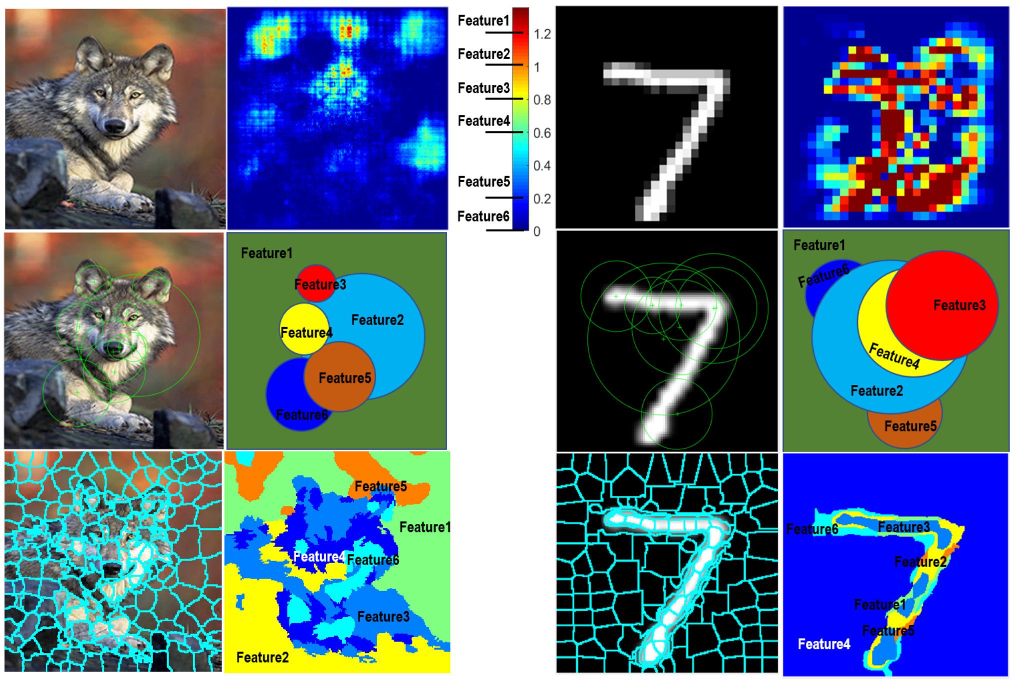


Figure 5.6: Examples of three feature extraction methods applied on the ImageNet (left) and MNIST (right) datasets, respectively. The top row: image segmentation via the saliency map generation method introduced in [60]. Middle row: feature extraction using the SIFT approach [41]. Bottom row: image partition using K-means clustering and superpixels [73].

partitioning method into disjoint sets of dimensions. In Figure 5.6 we illustrate three distinct feature extraction procedures on a colour image from the ImageNet dataset and a grey-scale image from the MNIST dataset. Though we work with image classifier networks, our approach is flexible and can be adapted to a range of feature partitioning methods.

The first technique for image segmentation is based on the saliency map generated from an image classifier such as a neural network. As shown Figure 5.6 (top row), the heat-map is produced by quantifying how sensitive each pixel is to the classification outcome of the network. By ranking these sensitivities, we separate the pixels into a few disjoint sets. The second feature extraction approach, shown in Figure 5.6 (middle row), is independent of any image classifier, but instead focuses on abstracting the invariant properties directly from the image. Here we show segmentation results from the SIFT method [41], which is invariant to

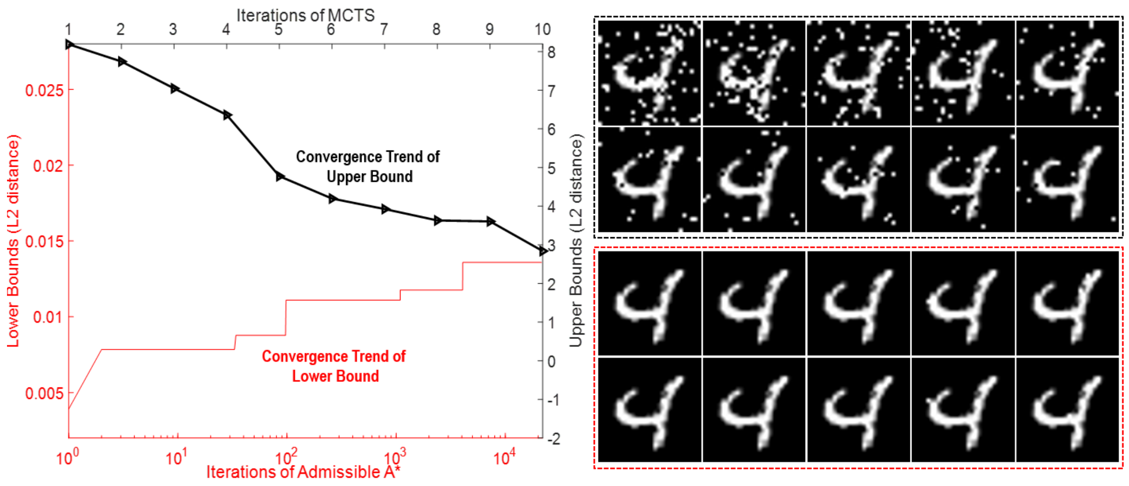


Figure 5.7: Convergence of *maximum safe radius* in a *cooperative* game with grey-box feature extraction of an MNIST image originally classified as “4”. Left: The convergence trends of the upper bound from MCTS and the lower bound from Admissible A* for the MSR problem. Right: The generated *adversarial* images while searching for the upper bound via MCTS, and lower boundary *safe* images while searching for the lower bound via Admissible A*.

image translation, scaling, rotation, and local geometric distortion. More details on how to adapt SIFT for safety verification on neural networks can be found in [76]. The third feature extraction method is based on superpixel representation, a dimensionality reduction technique widely applied in various computer vision applications. Figure 5.6 (bottom row) demonstrates an example of how to generate superpixels (i.e., the pixel clusters marked by the green grids) using colour features and K-means clustering [73].

5.4.2 Convergence Analysis of the Upper and Lower Bounds

We demonstrate convergence of the bound computation for the maximum safe radius and feature robustness problems, evaluated on standard benchmark datasets MNIST, CIFAR-10, and GTSRB. The architectures of the corresponding trained neural networks as well as their accuracy rates can be found in Appendix B.2.1.

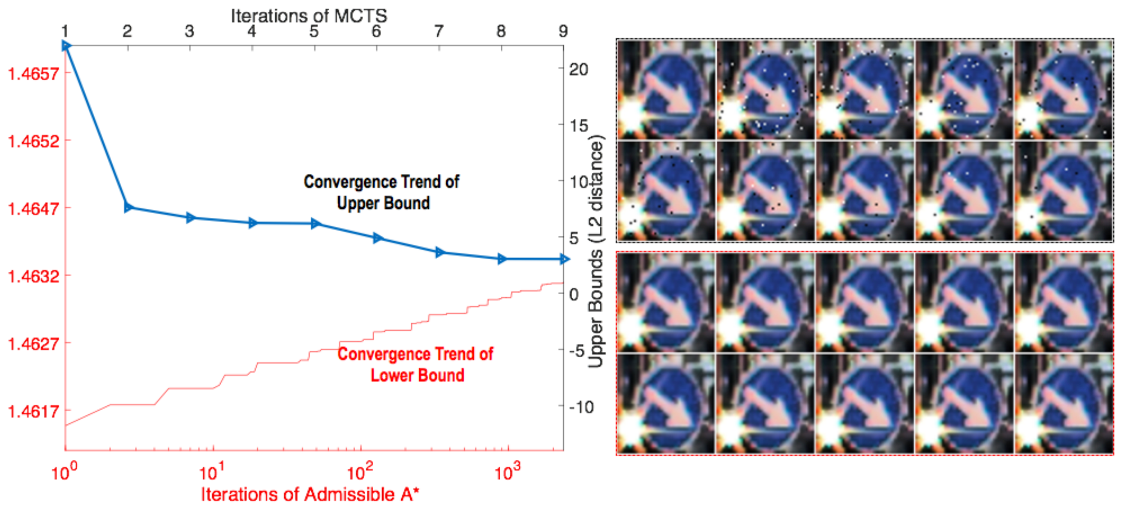
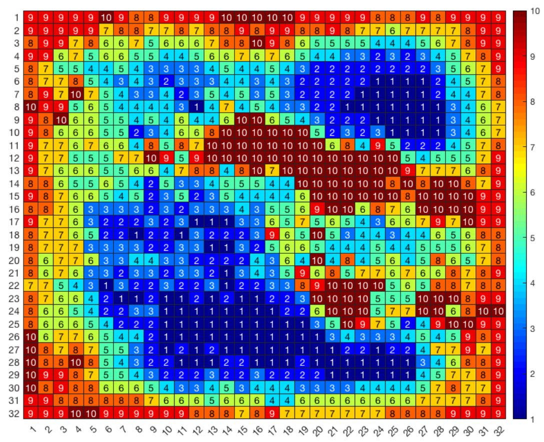


Figure 5.8: Convergence of *maximum safe radius* in a *cooperative* game with grey-box feature extraction of a GTSRB image originally classified as "keep right". Left: The convergence trends of the upper bound from MCTS and the lower bound from Admissible A* for the MSR problem. Right: The generated *adversarial* images while searching for the upper bound via MCTS, and lower boundary *safe* images while searching for the lower bound via Admissible A*.

Convergence of MSR in a Cooperative Game

First, we illustrate convergence of MSR in a *cooperative* game on the MNIST and GTSRB datasets. For the MNIST image (index 67) in Figure 5.7, the black line denotes the descending trend of the upper bound $\mathsf{u}_{\mathtt{MSR}}$, whereas the red line indicates the ascending trend of the lower bound $\mathsf{l}_{\mathtt{MSR}}$. Intuitively, after a few iterations, the upper bound (i.e., minimum distance to an adversarial example) is 2.84 wrt the L^2 metric, and the absolute safety (i.e., lower bound) is within radius 0.012 from the original image. The right-hand side of Figure 5.7 includes images produced by intermediate iterations, with *adversarial* images generated by MCTS shown in the two top rows, and *safe* images computed by Admissible A* in the bottom rows. Similarly, Figure 5.8 displays the converging upper and lower bounds of MSR in a *cooperative* game on a GTSRB image (index 19).

As for the computation time, each MCTS iteration updates the upper bound $\mathsf{u}_{\mathtt{MSR}}$ and typically takes minutes; each Admissible A* iteration further expands the game tree and updates the lower bound $\mathsf{l}_{\mathtt{MSR}}$ whenever applicable. The running


(a) CIFAR-10 "ship" image.

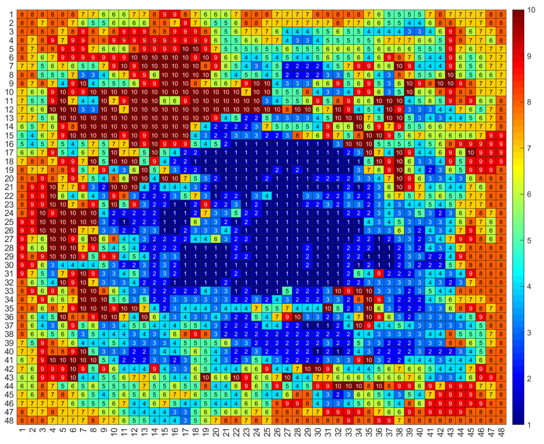

(b) GTSRB "speed limit 70 mph" image.

Figure 5.9: Illustration of the 10 features using the grey-box feature extraction procedure: cells with the same colour indicate the same feature, and the number in each cell represents the featureID. That is, Feature1 in deep blue has the most salient impact, whereas Feature10 in deep red is the least influential. (a) Features of the CIFAR-10 "ship" image (32×32) in Figure 5.10. (b) Features of the GTSRB "speed limit 70 mph" image (48×48) in Figure 5.11.

times for the iterations of the Admissible A* vary: initially it takes minutes but this can increase to hours when the tree is larger.

Convergence of $\mathtt{FR}_\Lambda$ in a Competitive Game

Next we demonstrate the convergence of $\mathtt{FR}_\Lambda$ in a *competitive* game on the CIFAR-10 and GTSRB datasets. Each iteration of MCTS or Alpha-Beta Pruning updates their respective bound with respect to a certain feature. Note that, in each MCTS iteration, upper bounds $\mathsf{u}_{\mathtt{MSR}}$ of all the features are improved and therefore the maximum among them, i.e., $\mathsf{u}_{\mathtt{FR}_\Lambda}$ of the image, is updated, whereas Alpha-Beta Pruning calculates $\mathsf{l}_{\mathtt{MSR}}$ of a feature in each iteration, and then compares and updates $\mathsf{l}_{\mathtt{FR}_\Lambda}$ with the computation progressing until all the features are processed.

For the CIFAR-10 image in Figure 5.10, the green line denotes the upper bound $\mathsf{u}_{\mathtt{FR}_\Lambda}$ and the red line denotes the lower bound $\mathsf{l}_{\mathtt{FR}_\Lambda}$. The "ship" image is partitioned into 10 features (see Figure 5.9(a)) utilising the grey-box extraction method. We observe that this saliency-guided image segmentation procedure captures the features

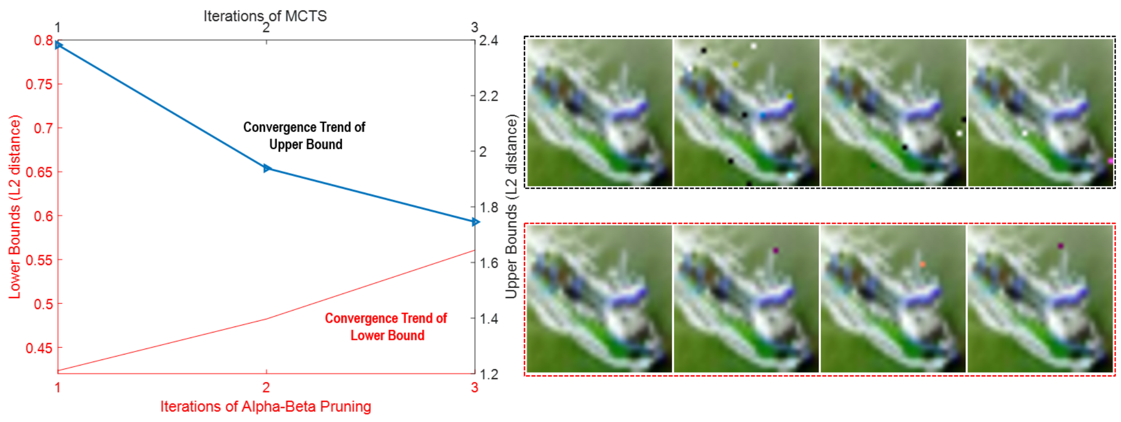


Figure 5.10: Convergence of *feature robustness* in a *competitive* game with grey-box feature extraction of a CIFAR-10 image originally classified as "ship". Left: The convergence trends of the upper bound from MCTS and the lower bound from Alpha-Beta Pruning for the $\mathtt{FR}_\Lambda$ problem. Right: The generated adversarial images while computing the upper bounds via MCTS, and lower bound images while computing the lower bounds via Alpha-Beta Pruning.

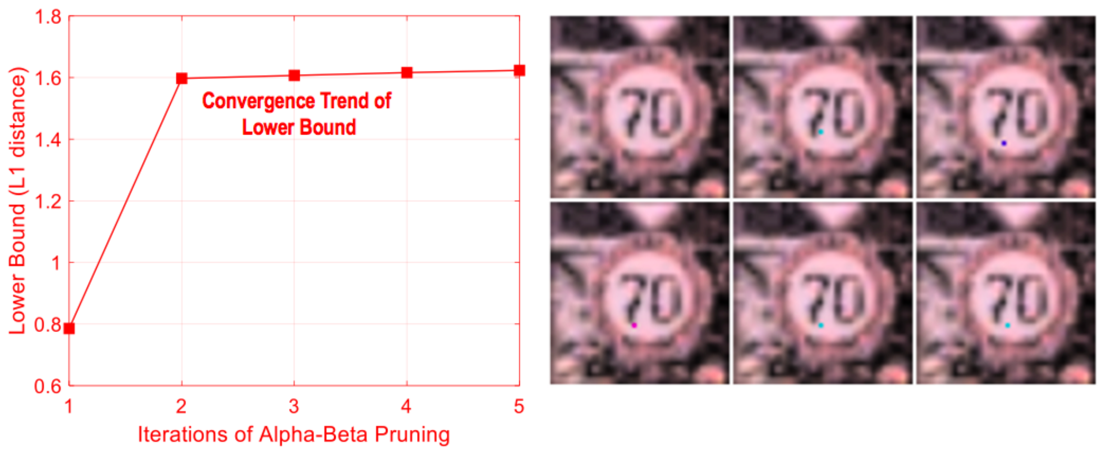


Figure 5.11: Convergence of *feature robustness* in a *competitive* game with the grey-box feature extraction of a GTSRB image originally classified as "speed limit 70 mph". Left: The convergence trends of the lower bound from Alpha-Beta Pruning for the $\mathtt{FR}_\Lambda$ problem. Right: The generated lower bound images while computing the lower bounds via Alpha-Beta Pruning.

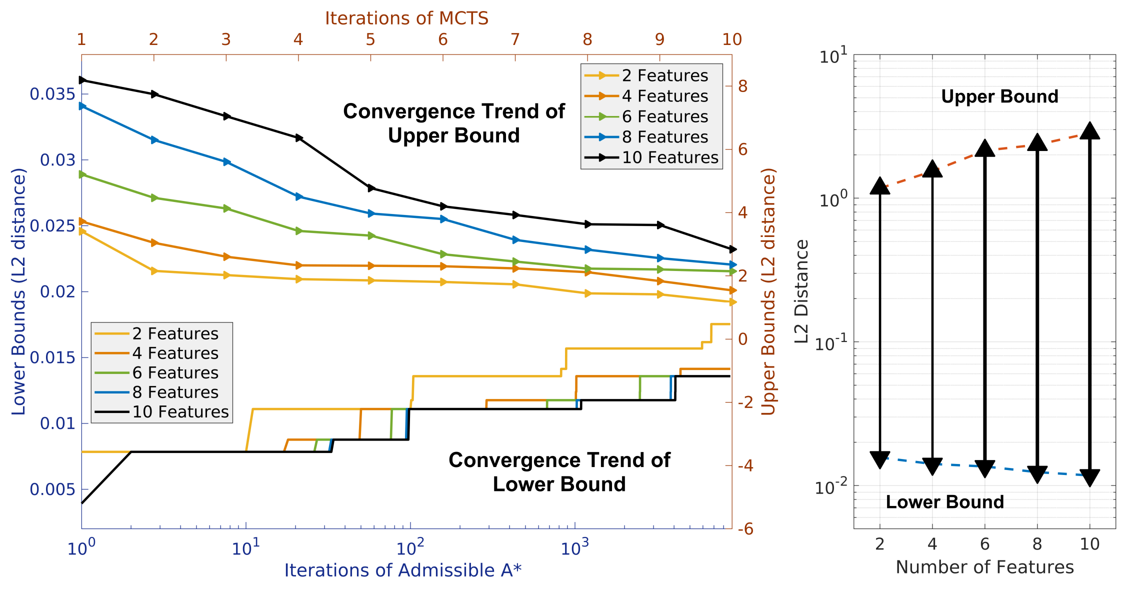
well, as in Figure 5.9(a) the most influential features (in blue) resemble the silhouette of the "ship". After 3 iterations, the algorithm indicates that, at L^2 distance of more than 1.75, all features are fragile, and if the L^2 distance is 0.48 there exists at least one robust feature. The right-hand side of Figure 5.10 shows several intermediate images produced, along with the converging $\mathsf{u}_{\mathtt{FR}_\Lambda}$ and $\mathsf{l}_{\mathtt{FR}_\Lambda}$. The top row exhibits the original image as well as the manipulated images with decreasing u_{FR}. For instance, after the 1st iteration, MCTS finds an adversary perturbed in Feature4 with L^2 distance 2.38, which means by far the most robust feature of this "ship" image is Feature4. (FeatureID is retrieved from the number in each cell of the image segmentation in Figure 5.9(a).) When the computation proceeds, the 2nd iteration updates $\mathsf{u}_{\mathsf{FR}_\Lambda}$ from 2.38 to 1.94, and explores the current most robust Feature8, which is again replaced by Feature9 after the 3rd iteration with lower distance 1.75. The bottom row displays the original image together with perturbations in each feature while $\mathsf{l}_{\mathsf{FR}_\Lambda}$ is increasing. It can be seen that Feature1, Feature2, and Feature3 need only one dimension change to cause image misclassification, and the lower bound Feature4 increases from 0.42 to 0.56 after three iterations.

For the *feature robustness* ($\mathtt{FR}_\Lambda$) problem, i.e., when Player $\mathtt{I}$ and Player $\mathtt{II}$ are competing against each other, apart from the previous CIFAR-10 case where Player $\mathtt{II}$ wins the game by generating an adversarial example with atomic manipulations in each feature, there is a chance that Player $\mathtt{I}$ wins, i.e., at least one robust feature exists. Figure 5.11 illustrates this scenario on the GTSRB dataset. Here Player $\mathtt{I}$ defeats Player $\mathtt{II}$ through finding at least one robust feature by MCTS, and thus the convergence trend of the upper bound $\mathsf{u}_{\mathsf{FR}_\Lambda}$ is not shown. As for the lower bound $\mathsf{l}_{\mathsf{FR}_\Lambda}$, Alpha-Beta Pruning enables Player $\mathtt{II}$ to manipulate a single pixel in Feature1 - Feature5 (see Figure 5.9(b)) so that adversarial examples are found. For instance, with L^1 distance above 0.79, Feature1 turns out to be fragile.

Here, each iteration of MCTS or Alpha-Beta Pruning is dependent on the size of feature partitions – for smaller partitions it takes seconds to minutes, whilst for larger partitions it can take hours. The running times are also dependent on the norm ball radius d. If the radius d is small, the computation can always terminate in minutes.

Scalability wrt Number of Input Dimensions

We now investigate how the increase in the number of dimensions affects the convergence of the lower and upper bounds. From the complexity analysis of the



(a) Convergence of lower and upper bounds. (b) Gap of bounds.

Figure 5.12: Analysis of convergence of upper and lower bounds of the *maximum safe radius* as the number of dimensions increases for the MNIST image in Figure 5.7, based on the L^2 norm. The increase in the number of features (2 to 10) corresponds to an increase in the number the input dimensions.

problems in Section 5.2.3, we know that the theoretical complexity is in PTIME with respect to the size of the game model, which is exponential with respect to the number of input dimensions.

We utilise the example in Figure 5.7, where the convergence of the upper bound $\mathsf{u}_{\mathtt{MSR}}$ and the lower bound $\mathsf{l}_{\mathtt{MSR}}$ in a cooperative game is exhibited on *all* the dimensions (pixels) of the MNIST image (index 67). We partition the image into 10 disjoint features using the grey-box extraction method, and gradually manipulate features, starting from those with fewer dimensions, to observe how the corresponding bound values $\mathsf{u}_{\mathtt{MSR}}$, $\mathsf{l}_{\mathtt{MSR}}$ are affected if we fix a time budget. To ensure fair comparison, we run the same number of expansions of the game tree, i.e., 10 iterations of MCTS, and 1000 iterations of Admissible A*, and plot the bound values $\mathsf{u}_{\mathtt{MSR}}$, $\mathsf{l}_{\mathtt{MSR}}$ thus obtained. Figure 6.11 shows the widening upper and lower bounds based on the L^2 norm with respect to 2 to 10 features of the image. It is straightforward to see that the conclusion also holds for the *feature robustness* problem.

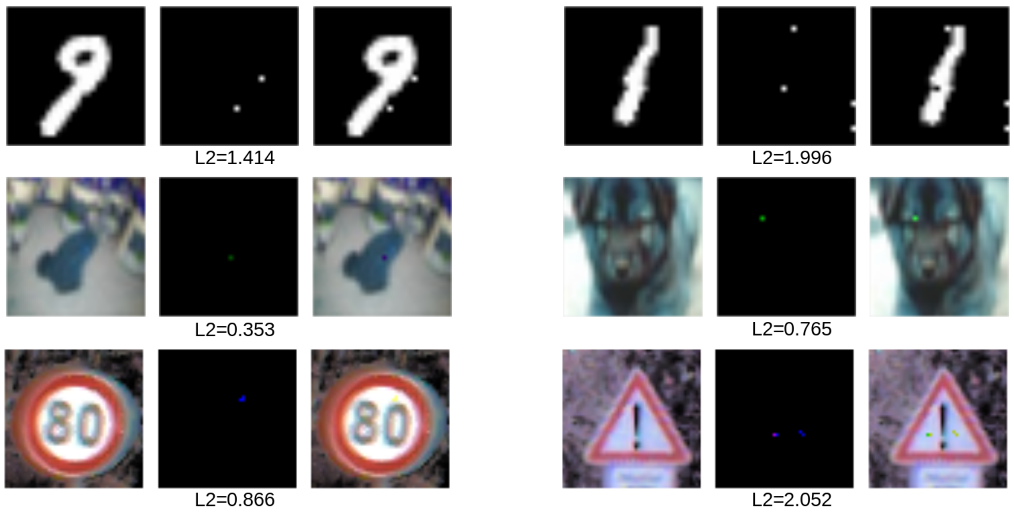


Figure 5.13: Examples of adversarial MNIST, CIFAR-10 and GTSRB images with slight perturbations based on the L^2 norm. Top: "9" misclassified into "8"; "1" misclassified into "3". Middle: "frog" misclassified into "dog"; "dog" misclassified into "cat". Bottom: "speed limit 80 mph" misclassified into "speed limit 60 mph"; "danger" misclassified into "pedestrian crossing".

5.4.3 Comparison with Existing Approaches in Adversarial Attacks

When the game is cooperative, i.e., for the *maximum safe radius* problem, we can have adversarial examples as by-products. In this regard, both MCTS and A* algorithm can be applied to generate adversarial examples. Note that, for the latter, we can take Inadmissible A* (i.e., the heuristic function can be inadmissible), as the goal is not to ensure the lower bound but to find adversarial examples. By proportionally enlarging the heuristic distance $heuristic(\boldsymbol{\alpha}')$ with a constant, we ask the algorithm to explore those tree nodes where an adversarial example is more likely to be found. Figure 5.13 displays some adversarial MNIST, CIFAR-10 and GTSRB images generated by DeepGame after manipulating a few pixels. More examples can be found in Figures B.1, B.2, and B.3.

We compare our tool DeepGame with several state-of-the-art approaches to search for adversarial examples: C&W [8], DeepTRE [60], DLV [28], SafeCV [76],

[1]Whilst DeepGame works on channel-level dimension of an image, in order to align with some tools that attack at pixel level the statistics are all based on the number of different pixels.

Table 5.1: Comparison between our tool DeepGame and several other tools on search for adversarial examples performed on the MNIST and CIFAR-10 datasets, based on the Hamming distance. Here DeepGame deploys the grey-box feature extraction method, and Inadmissible A* algorithm. We set a ten-minute time constraint and evaluate on correctly classified images and the produced adversarial examples.

Hamming Distance	MNIST				CIFAR-10[1]			
	Distance		Time (s)		Distance		Time (s)	
	mean	std	mean	std	mean	std	mean	std
DeepGame	**6.11**	**2.48**	**4.06**	**1.62**	**2.86**	**1.97**	**5.12**	**3.62**
C&W	7.07	4.91	17.06	1.80	3.52	2.67	15.61	5.84
DeepTRE	10.85	6.15	0.17	0.06	2.62	2.55	0.25	0.05
DLV	13.02	5.34	180.79	64.01	3.52	2.23	157.72	21.09
SafeCV	27.96	17.77	12.37	7.71	9.19	9.42	26.31	78.38
JSMA	33.86	22.07	3.16	2.62	19.61	20.94	0.79	1.15

and JSMA [53]. More specifically, we train neural networks on two benchmark datasets, MNIST and CIFAR-10, and calculate the distance between the adversarial image and the original image based on the Hamming distance. The original images, preprocessed to be within the bound $[0, 1]$, are the first 1000 images of each testing set. Apart from a ten-minute time constraint, we evaluate on correctly classified images and their corresponding adversarial examples. This is because some tools regard misclassified images as adversarial examples and record zero-value distance while other tools do not, which would result in unfair comparison.

Table 5.1 demonstrates the statistics. Figures B.1 and B.2 include adversarial examples found by these tools. Model architectures and the parameter settings for these tools can be found in Appendix B.2.

5.5 Summary

In this chapter, we present a two-player turn-based game framework for the verification of deep neural networks with provable guarantees. In particular, we tackle two problems, *maximum safe radius* and *feature robustness*, which essentially correspond to the absolute (pixel-level) and relative (feature-level) safety of a network against adversarial manipulations. Our framework can deploy various feature extraction or image segmentation approaches, including the saliency-guided grey-box mechanism, and the feature-guided black-box procedure. We develop a software tool DeepGame, and demonstrate its applicability on state-of-the-art networks and dataset benchmarks. Our experiments exhibit converging upper and lower bounds, and are competitive compared to existing approaches in searching for adversarial examples.

6 Robustness of Deep Neural Networks on Videos

Contents

No matter whether we are studying the *maximum safe radius* problem, or the *feature robustness* problem, the previous Chapters 4 and 5 evaluate the robustness of neural networks on images, while in this chapter we adapt the methods to time-series data, e.g., *video* inputs. This is due to the fact that human decision-making is more of a dynamic progress rather than a static one – we keep perceiving the surrounding environment and then developing the corresponding strategies. We remark that video classification is more challenging than image recognition because, apart from the *spatial* features on each frame, extracted by the convolutional networks, videos also consist of the *temporal* dynamics between adjacent frames, captured by the recurrent networks. Moreover, adversarial perturbations of videos include more variants than pixel manipulations in images, such as frame loss/repetition, brightness change, and camera occlusion. Therefore, in this chapter we continue using the L^p norms where $p \geq 1$ to measure the distance between the original input and the perturbed one.

Regarding the methodology behind the robustness guarantees for deep neural networks on videos, in general we adapt the game-based approach proposed in Chapter 5, and highlight the key differences. To start with, instead of computing the maximum safe radius of an image input, here we extend to the concept of the *maximum safe radius with respect to optical flow*, so that both the spatial and temporal features in a video can be captured by the flow. Consequently, in a two-player turn-based game, this time Player I no longer chooses image partitions but optical flows, and Player II determines adversarial perturbations within the chosen flow. We will still aim to compute upper and lower bounds, although for videos we propose a gradient-based search algorithm to compute the upper bounds. Finally, in experiments we add recurrent layers into the networks to deal with the time-series characteristics and evaluate them on a video dataset, which has input dimensions considerably greater than the image benchmarks used in previous chapters.

This chapter is organised as follows. In Section 6.1, we extend the formulation of the maximum safe radius to that with respect to optical flow, and reduce the problem into finite optimisation via Lipschitz continuity. Subsequently, we reuse the game-based approximate verification framework in Section 6.2, where this time Player I selects optical flows and Player II determines perturbations within the chosen flow. In Section 6.3, we design a gradient-based search algorithm to compute the upper bounds and reuse the admissible A* algorithm to generate the lower bounds. Finally, we report the experimental results in Section 6.4 produced by our tool DeepVideo.

6.1 Robustness on Videos

In this section, we formulate the robustness problem and provide an approximation with provable guarantees. We reuse the notions defined in Chapter 4, and extend the input from an image $\boldsymbol{\alpha}$ to a video $\boldsymbol{v}$. That is, we have the norm ball $\mathsf{Ball}(\boldsymbol{v}, L^p, d)$, the local robustness property $\textsc{Robust}(\mathcal{N}, \boldsymbol{v}, L^p, d)$, and the maximum safe radius $\texttt{MSR}(\mathcal{N}, \boldsymbol{v}, L^p, d)$, which are exactly the same as in Chapter 4, except the input type is now a video.

6.1.1 Maximum Safe Radius with respect to Optical Flow

In existing work that evaluate a network's robustness over images, it is common to manipulate each image at pixel- or channel-level, and then compute the distance between the perturbed and original inputs. However, as we deal with time-series inputs, i.e., *videos*, instead of manipulating directly on each individual frame, we impose perturbation on each *optical flow* that is extracted from every pair of adjacent frames, so that both spatial features on frames and temporal dynamics between frames can be captured. We define optical flow as follows.

Definition 6.1 (Optical Flow)**.** *Given an input video $\boldsymbol{v}$ with number l of frames, i.e., $\boldsymbol{v} = \{\mathcal{F}_1, \ldots, \mathcal{F}_t, \ldots, \mathcal{F}_l\}, t \in [1, l], t \in \mathbb{N}^+$, the* optical flow *extraction function $f : \mathcal{F}_t, \mathcal{F}_{t+1} \mapsto \mathtt{op}_t$ maps every two adjacent frames $\mathcal{F}_t, \mathcal{F}_{t+1}$ into an optical flow $\mathtt{op}_t$. Then, for the video $\boldsymbol{v}$, a* sequence of optical flows *can be extracted, i.e., $\mathsf{OP}(\boldsymbol{v}) = \{\mathtt{op}_1, \ldots, \mathtt{op}_t, \ldots, \mathtt{op}_{l-1}\}, t \in [1, l-1], t \in \mathbb{N}^+$.*

We remark that the distance between the optical flow sequences of two videos, denoted as $\|\mathsf{OP}(\boldsymbol{v}) - \mathsf{OP}(\boldsymbol{v}')\|_p$, can be measured similarly to the distance of two videos $\|\boldsymbol{v} - \boldsymbol{v}'\|_p$ by the $L^p, p \in \{1, 2, \infty\}$ norms in a standard way, as they are essentially tensors.

Then, to study the crafting of adversarial examples, we construct *manipulations* on the optical flow to obtain perturbed inputs. Note that if the input values are bounded, e.g., $[0, 255]$ or $[0, 1]$, then the perturbed inputs need to be restricted to be within the bounds.

Definition 6.2 ((Atomic) Optical Flow Manipulation)**.** *Given an input $\boldsymbol{v}$ with a set of optical flow $\mathsf{OP}(\boldsymbol{v})$, an instruction function $\psi : \mathbb{R} \to \mathbb{N}$, and a manipulation magnitude τ, we define the* input manipulation *operations*

$$\delta_{\psi,\tau}(\mathtt{op}_t)(i) = \begin{cases} \mathtt{op}_t[i] + \psi(i) \cdot \tau, & \text{if } i \in [1, w \times h], i \in \mathbb{N}^+ \\ \mathtt{op}_t[i], & \text{otherwise} \end{cases} \tag{6.1}$$

where w, h denote the width and height of $\boldsymbol{v}$. Specifically, when $\psi : \mathbb{R} \to \{+1, -1\}$, we say the manipulation is atomic*, denoted as $\delta_{\theta,\tau}$.*

Moreover, after remapping the manipulated flow back to the original frame, we obtain a perturbed new frame, i.e., $f' : \mathcal{F}_t, \delta_{\psi,\tau}(\mathtt{op}_t) \to \mathcal{F}'_{t+1}$, and the manipulated flow set, $f' : \boldsymbol{v}, \delta_{\psi,\tau}(\mathsf{OP}(\boldsymbol{v})) \to \boldsymbol{v}'$, maps to a new video with the perturbation. Intuitively, given a video input and a neural network, we apply some optical flow extraction method to generate the sequence of flows. The flow set is then modified by manipulations, and subsequently restored to the original input to obtain a perturbed video, which is classified by the network to see if it is an adversarial example. To this end, we compute the distance from $\delta_{\psi,\tau}(\mathsf{OP}(\boldsymbol{v}))$ to $\mathsf{OP}(\boldsymbol{v})$ instead of that from $\boldsymbol{v}'$ to $\boldsymbol{v}$ because the former reflects both spatial and temporal manipulations simultaneously. That is, we compute the *maximum safe radius with respect to optical flow*, denoted as $\mathtt{MSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d)$, such that $\mathcal{N}(\boldsymbol{v}') \neq \mathcal{N}(\boldsymbol{v})$.

6.1.2 Approximation based on Finite Optimisation

By manipulating optical flow, we explore the existence of potential adversarial examples $\boldsymbol{v}'$ of input $\boldsymbol{v}$. The $\textsc{Robust}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d)$ and $\mathtt{MSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d)$ problems cover the set of all the input points, which are essentially infinite. In this section, we utilise the fact that the networks studied in this thesis are *Lipschitz continuous* to discretise the neighbourhood space of an optical flow set, i.e., transform the infinite number of points in the norm ball into a finite number on the grid. First, based on the definitions of optical flow and input manipulation, we transform the `MSR` problem into the following *finite maximum safe radius* problem.

Definition 6.3 (Finite Maximum Safe Radius). *Given an input $\boldsymbol{v}$, and a manipulation function $\delta_{\psi,\tau}$, let $\boldsymbol{v}' = f'(\boldsymbol{v}, \delta_{\psi,\tau}(\mathsf{OP}(\boldsymbol{v})))$ denote the perturbed input, then the* finite maximum safe radius with respect to optical flow *is*

$$\texttt{FMSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d, \tau) = \min_{\mathsf{op}_t \in \mathsf{OP}(\boldsymbol{v})} \min_{\theta \in \psi} \{ \|\mathsf{OP}(\boldsymbol{v}) - \delta_{\psi,\tau}(\mathsf{OP}(\boldsymbol{v}))\|_p \mid \delta_{\psi,\tau}(\mathsf{OP}(\boldsymbol{v})) \in \mathsf{Ball}(\mathsf{OP}(\boldsymbol{v}), L^p, d) \ \textit{s.t.}\ \mathcal{N}(\boldsymbol{v}') \neq \mathcal{N}(\boldsymbol{v}) \}. \quad (6.2)$$

If $\boldsymbol{v}'$ does not exist in Ball, *we let* $\texttt{FMSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d, \tau) = d + \epsilon$.

Intuitively, we aim to find a set of manipulations $\theta \in \psi$ to impose on a set of optical flows $\mathsf{op}_t \in \mathsf{OP}(\boldsymbol{v})$, such that the distance between the flow sets is minimal, and after the remapping procedure the perturbed input $\boldsymbol{v}'$ is an adversarial example. Considering that, within a norm ball Ball, the set of manipulations is finite for a fixed magnitude τ, the `FMSR` problem only needs to explore a finite number of the 'grid' points. To achieve this, we let $\overline{g}$ be a τ-grid point such that $|\overline{g} - \mathsf{OP}(\boldsymbol{v})| = n \times \tau$, and $\Gamma(\mathsf{OP}(\boldsymbol{v}), L^p, d)$ be the set of τ-grid points whose corresponding inputs are in Ball. Note that all the τ-grid points are reachable from each other via manipulation. By selecting a proper τ, we ensure that the optical flow space can be covered by small sub-spaces. That is,

$$\mathsf{Ball}(\mathsf{OP}(\boldsymbol{v}), L^p, d) \subseteq \bigcup_{\overline{g} \in \Gamma} \mathsf{Ball}(\overline{g}, L^p, \frac{1}{2}\tilde{d}(L^p, \tau)), \quad (6.3)$$

where the grid width $\tilde{d}(L^p, \tau)$ is $|\mathrm{D}|\tau$ for L^1, $\sqrt{|\mathrm{D}|\tau^2}$ for L^2, and τ for L^∞.

To this point, we can use `FMSR` to approximate `MSR` within the error bounds in Theorem 11, as illustrated in Figure 6.1. Intuitively, this is similar to Lemmas 3 and 5 in Chapter 5, with the major difference being that in this case the discretisation method is imposed upon the space of optical flow instead of the input space. In order to reduce the verification problem to manipulating just the grid points, we

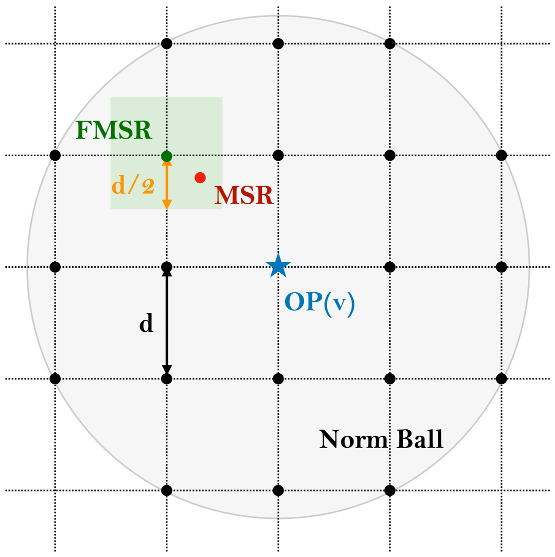


Figure 6.1: Provable guarantees for *maximum safe radius* w.r.t. optical flow via a τ-grid discretisation of the norm ball. Here, $\mathsf{OP}(\boldsymbol{v})$ in blue is the original optical flow set, around which the grey area within the black circle is the norm ball Ball with grid width $\tilde{d}(L^p, \tau)$. The found optimum `FMSR` (green), as a τ-grid point, can bound the true optimum `MSR` (red) within $\frac{1}{2}\tilde{d}(L^p, \tau)$ distance.

need to bound the output behaviour of the network within each grid cell, which can be achieved by Lipschitz continuity.

Theorem 11 (Error Bounds)**.** *Given a manipulation magnitude τ, the optical flow space can be discretised into a set of τ-grid points, and* `MSR` *can be approximated as*

$$\begin{aligned}\mathtt{FMSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d, \tau) - \frac{1}{2}\tilde{d}(L^p, \tau) \\ \leq \mathtt{MSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d) \leq \mathtt{FMSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d, \tau). \quad (6.4)\end{aligned}$$

Then, the problem is to determine the manipulation magnitude τ. Note that, in order to make sure each τ-grid point $\overline{g}$ covers all the possible manipulation points in its neighbourhood, we compute the largest τ. We now show that τ can be obtained via *Lipschitz continuity*. For a network $\mathcal{N}$ which is Lipschitz continuous at input $\boldsymbol{v}$, given Lipschitz constant $\mathsf{Lip}_c, c \in C$, for each class, we have

$$\tilde{d}'(L^p, \tau) \leq \frac{\min_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} \{\mathcal{N}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v})) - \mathcal{N}(\boldsymbol{v}, c)\}}{\max_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{v})} + \mathsf{Lip}_c)}. \quad (6.5)$$

Below we prove how this quantity is computed. Recall that in Chapter 5 we introduce the concept of the *minimum confidence margin* (Definition 5.8). Here we apply it to the domain of video inputs. Intuitively, it is the discrepancy between the maximum confidence of $\boldsymbol{v}$ being classified as c and the next largest confidence of $\boldsymbol{v}$ being classified as c'.

Definition 6.4 (Minimum Confidence Margin)**.** *Given a network $\mathcal{N}$, an input $\boldsymbol{v}$, and a class c, we define the* minimum confidence margin *as*

$$\mathsf{Margin}(\boldsymbol{v}, c) = \min_{c' \in C, c' \neq c} \{\mathcal{N}(\boldsymbol{v}, c) - \mathcal{N}(\boldsymbol{v}, c')\}. \tag{6.6}$$

By utilising the above concept and Lipschitz continuity, we determine the value of manipulation magnitude τ so that every grid point can 'represent' it surrounding subspace. When Expression 6.5 is satisfied, Theorem 11 holds.

Proof. For any input $\boldsymbol{v}'$ whose optical flow set is in the subspace of a grid point $\overline{g}$, and the input $\boldsymbol{v}$ corresponding to this optical flow set $\overline{g}$, we have

$$\begin{aligned}
&\mathsf{Margin}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v})) - \mathsf{Margin}(\boldsymbol{v}', \mathcal{N}(\boldsymbol{v}))\\
&= \min_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} \{\mathcal{N}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v})) - \mathcal{N}(\boldsymbol{v}, c)\} - \min_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} \{\mathcal{N}(\boldsymbol{v}', \mathcal{N}(\boldsymbol{v})) - \mathcal{N}(\boldsymbol{v}', c)\}\\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} \{\mathcal{N}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v})) - \mathcal{N}(\boldsymbol{v}, c) - \mathcal{N}(\boldsymbol{v}', \mathcal{N}(\boldsymbol{v})) + \mathcal{N}(\boldsymbol{v}', c)\}\\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} \{|\mathcal{N}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v})) - \mathcal{N}(\boldsymbol{v}', \mathcal{N}(\boldsymbol{v}))| + |\mathcal{N}(\boldsymbol{v}', c) - \mathcal{N}(\boldsymbol{v}, c)|\}\\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} \mathsf{Lip}_{\mathcal{N}(\boldsymbol{v})} \cdot \|\boldsymbol{v} - \boldsymbol{v}'\|_p + \mathsf{Lip}_c \cdot \|\boldsymbol{v} - \boldsymbol{v}'\|_p\\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{v})} + \mathsf{Lip}_c) \cdot \|\boldsymbol{v} - \boldsymbol{v}'\|_p\\
&\leq \max_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{v})} + \mathsf{Lip}_c) \cdot \tilde{d}'(L^p, \tau)
\end{aligned} \tag{6.7}$$

Now, since the optical flow set of $\boldsymbol{v}'$ is in the subspace of $\overline{g}$, we need to ensure that no class change occurs between $\boldsymbol{v}$ and $\boldsymbol{v}'$. That is, $\mathsf{Margin}(\boldsymbol{v}', \mathcal{N}(\boldsymbol{v})) \geq 0$, which means $\mathsf{Margin}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v})) - \mathsf{Margin}(\boldsymbol{v}', \mathcal{N}(\boldsymbol{v})) \leq \mathsf{Margin}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v}))$. Therefore, we have

$$\max_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{v})} + \mathsf{Lip}_c) \cdot \tilde{d}'(L^p, \tau) \leq \mathsf{Margin}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v})). \tag{6.8}$$

And as $\overline{g}$ is the grid point, the minimum confidence margin for its corresponding

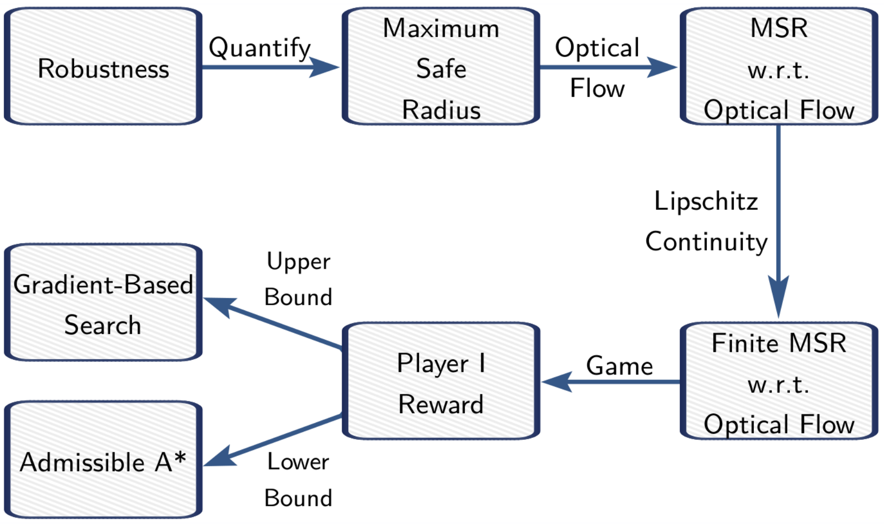


Figure 6.2: A game-based robustness verification approach for the *maximum safe radius* problem of deep neural networks on videos.

input $\boldsymbol{v}$ can be computed. Finally, we replace $\mathsf{Margin}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v}))$ with its definition, then we have

$$\tilde{d}'(L^p, \tau) \leq \frac{\min\limits_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} \{\mathcal{N}(\boldsymbol{v}, \mathcal{N}(\boldsymbol{v})) - \mathcal{N}(\boldsymbol{v}, c)\}}{\max\limits_{c \in C, c \neq \mathcal{N}(\boldsymbol{v})} (\mathsf{Lip}_{\mathcal{N}(\boldsymbol{v})} + \mathsf{Lip}_c)}. \tag{6.9}$$

□

Remark 3. *We remark that, while $\tilde{d}'(L^p, \tau)$ is with respect to input $\boldsymbol{v}$ and $\tilde{d}(L^p, \tau)$ is with respect to the flow set $\mathsf{OP}(\boldsymbol{v})$, the relation between them, and similarly that between f and f', is dependent on the optical flow extraction method used. As this is not the main focus of this thesis, we do not expand on this topic.*

6.2 A Game-Based Robustness Verification Approach

In this section, we show that the finite optimisation problem `FMSR` of Definition 6.3 can be reduced to the computation of Player `I`'s reward when taking the optimal strategy in a game-based setting. To this end, we adapt the game-based approach proposed in Chapter 5 for robustness evaluation of convolutional networks on images. The overall workflow is illustrated in Figure 6.2. Here we highlight the differences between these two approaches: (1) instead of computing the maximum safe radius of an image, this chapter extends to that with respect to optical flow; (2) to enable finite optimisation, Lipschitz continuity is still applied but to discretise the flow space rather than the input space; (3) a novel gradient-based search algorithm is proposed to compute the upper bounds, utilising the spatial features extracted from individual frames.

6.2.1 Problem Solving as A Two-Player Turn-Based Game

We define a two-player turn-based game, in which Player I chooses which optical flow to perturb, and Player II then imposes atomic manipulations of the pixels within the selected flow. Note that, in this chapter, we primarily address the maximum safe radius problem, therefore both player share the same objective – to minimise the distance to an adversarial example. In other words, the game is *cooperative*.

Definition 6.5 (Game). *Given an input* $\boldsymbol{v}$ *and its optical flow set* $\mathsf{OP}(\boldsymbol{v})$*, we let* $\mathcal{G}(\mathcal{N}, \boldsymbol{v}, L^p, d) = (S \cup (S \times \mathsf{OP}(\boldsymbol{v})), s_0, \{T_{\mathtt{I}}, T_{\mathtt{II}}\}, L)$ *be* a game model*, where*

- $S \cup (S \times \mathsf{OP}(\boldsymbol{v}))$ *denotes* the set of game states*, in which* S *is the set of* Player I*'s states whereas* $S \times \mathsf{OP}(\boldsymbol{v})$ *is the set of* Player II*'s states. Each* $s \in S$ *corresponds to an optical flow set* $\mathsf{OP}(s)$ *in the norm ball* $\mathsf{Ball}(\mathsf{OP}(\boldsymbol{v}), L^p, d)$.
- $s_0 \in S$ *is* the initial state *such that* $\mathsf{OP}(s_0)$ *corresponds to the original optical flow set* $\mathsf{OP}(\boldsymbol{v})$.
- $T_{\mathtt{I}} : S \times \mathsf{OP}(\boldsymbol{v}) \to S \times \mathsf{OP}(\boldsymbol{v})$ *is* Player I*'s* transition relation *defined as*

$$T_{\mathtt{I}}(s, \mathsf{op}_t) = (s, \mathsf{op}_t), \tag{6.10}$$

  and $T_{\mathtt{II}} : (S \times \mathsf{OP}(\boldsymbol{v})) \times \psi \to S$ *is* Player II*'s* transition relation

$$T_{\mathtt{II}}((s, \mathsf{op}_t), \theta) = \delta_{\theta,\tau}(\mathsf{op}_t), \tag{6.11}$$

where $\delta_{\theta,\tau}$ is the atomic manipulation of Definition 6.2. Intuitively, in a game state s, Player I *selects an optical flow* op_t *of* $\mathsf{OP}(s)$ *and enters into a* Player II*'s state* (s, op_t)*, where* Player II *then chooses an atomic manipulation* $\delta_{\theta,\tau}$ *on* op_t*.*

- $L : S \cup (S \times \mathsf{OP}(\boldsymbol{v})) \to C$ *is* the labelling function *that assigns each game state's corresponding input to a class* $\mathcal{N}(f'(\boldsymbol{v}, \mathsf{OP}(s)))$*.*

To compute FMSR of Definition 6.3, we let the game $\mathcal{G}$ be *cooperative*. When it proceeds, two players take turns - Player I employs a strategy σ_{I} to select optical flow, then Player II employs a strategy σ_{II} to determine atomic manipulations - thus forming a path ρ, which is a sequence $s_0\sigma_{\mathrm{I}}s_1\sigma_{\mathrm{II}}s_2\cdots$. Formally, we define the strategy of the game as follows.

Definition 6.6 (Strategy)**.** *Let $Path_{\mathrm{I}}^F$ be a set of finite paths ending in* Player I*'s state, and $Path_{\mathrm{II}}^F$ be a set of finite paths ending in* Player II*'s state, we define a* strategy profile $\sigma = (\sigma_{\mathrm{I}}, \sigma_{\mathrm{II}})$*, such that* $\sigma_{\mathrm{I}} : Path_{\mathrm{I}}^F \to \mathcal{D}(\mathsf{OP}(\boldsymbol{v}))$ *of* Player I *maps a finite path to a distribution over next actions, and similarly* $\sigma_{\mathrm{II}} : Path_{\mathrm{II}}^F \to \mathcal{D}(\psi)$ *for* Player II.

Intuitively, by imposing atomic manipulations in each round, the game searches for potential adversarial examples with increasing distance to the original optical flow. Given ρ, let $\boldsymbol{v}'_\rho = f'(\boldsymbol{v}, last(\rho))$ denote the input corresponding to the last state of ρ, and $\mathsf{OP}(\boldsymbol{v}'_\rho)$ denote its optical flow set, we write the *termination condition*

$$tc(\rho) \equiv (\mathcal{N}(\boldsymbol{v}'_\rho) \neq \mathcal{N}(\boldsymbol{v})) \vee (\left\|\mathsf{OP}(\boldsymbol{v}'_\rho) - \mathsf{OP}(\boldsymbol{v})\right\|_p > d), \tag{6.12}$$

which means the game is in a state whose corresponding input is either classified differently or outside the norm ball. In order to quantify the distance accumulated along a path, we define a reward function as follows. Intuitively, the reward is the distance to the original optical flow if an adversarial example is found, and otherwise it is the weighted summation of the rewards of its children on the game tree.

Definition 6.7 (Reward). *Give a strategy profile $\sigma = (\sigma_{\mathtt{I}}, \sigma_{\mathtt{II}})$, and a finite path ρ, we define a* reward *function*

$$R(\sigma,\rho) = \begin{cases} \left\|\mathsf{OP}(\boldsymbol{v}'_\rho) - \mathsf{OP}(\boldsymbol{v})\right\|_p, & \text{if } tc(\rho), \rho \in Path^F_{\mathtt{I}} \\ \sum_{\mathsf{op}_t \in \mathsf{OP}(\boldsymbol{v})} \sigma_{\mathtt{I}}(\rho)(\mathsf{op}_t) \cdot R(\sigma, \rho T_{\mathtt{I}}(last(\rho), \mathsf{op}_t)), & \text{if } \neg tc(\rho), \rho \in Path^F_{\mathtt{I}} \\ \sum_{\theta \in \psi} \sigma_{\mathtt{II}}(\rho)(\theta) \cdot R(\sigma, \rho T_{\mathtt{II}}(last(\rho), \theta)), & \text{if } \rho \in Path^F_{\mathtt{II}} \end{cases}, \tag{6.13}$$

where $\sigma_{\mathtt{I}}(\rho)(\mathsf{op}_t)$ is the probability of Player I *choosing optical flow op_t along ρ, and $\sigma_{\mathtt{II}}(\rho)(\theta)$ is the probability of* Player II *choosing atomic manipulation $\delta_{\theta,\tau}$ along ρ. Also, $\rho T_{\mathtt{I}}(last(\rho), \mathsf{op}_t)$ and $\rho T_{\mathtt{II}}(last(\rho), \theta)$ are the resulting paths of* Player I, Player II *applying $\sigma_{\mathtt{I}}, \sigma_{\mathtt{II}}$, respectively. Essentially, it is adding to ρ a new state after transition.*

6.2.2 Robustness Guarantees

We now confirm that the game can return the optical value of the reward function as the solution to the `FMSR` problem.

Theorem 12 (Guarantees). *Given an input $\boldsymbol{v}$, a game model $\mathcal{G}(\mathcal{N}, \boldsymbol{v}, L^p, d)$, and an optimal strategy profile $\sigma = (\sigma_{\mathtt{I}}, \sigma_{\mathtt{II}})$, the* finite maximum safe radius *problem is to minimise the reward of initial state s_0 based on σ, i.e.,* $\mathtt{FMSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d, \tau) = \min R(\sigma, s_0)$.

Proof. On one hand, we demonstrate that $\|\mathsf{OP}(\boldsymbol{v}') - \mathsf{OP}(\boldsymbol{v})\|_p \geq R(\sigma, s_0)$ for any optical flow set $\mathsf{OP}(\boldsymbol{v}')$ as a τ-grid point, such that $\mathsf{OP}(\boldsymbol{v}') \in \mathsf{Ball}(\mathsf{OP}(\boldsymbol{v}), L^p, d)$ and its corresponding input is an adversarial example. Intuitively, it means that Player I's reward from the game $\mathcal{G}$ on the initial state s_0 is no greater than the L^p distance to any τ-grid manipulated optical flow set. That is, the reward value $R(\sigma, s_0)$, once computed, is a lower bound of the optimisation problem $\mathtt{FMSR}(\mathcal{N}, \mathsf{OP}(\boldsymbol{v}), L^p, d, \tau)$. Note that the reward value can be obtained as every τ-grid point can be reached by some game play, i.e., a sequence of atomic manipulations.

On the other hand, from the termination condition $tc(\rho)$ of the game, we observe that, for some $\mathsf{OP}(\boldsymbol{v}')$, if $R(\sigma, s_0) \leq \|\mathsf{OP}(\boldsymbol{v}') - \mathsf{OP}(\boldsymbol{v})\|_p$ holds, then there must exist some other $\mathsf{OP}(\boldsymbol{v}'')$ such that $R(\sigma, s_0) = \|\mathsf{OP}(\boldsymbol{v}'' - \mathsf{OP}(\boldsymbol{v}))\|_p$. Therefore, we have that $R(\sigma, s_0)$ is the minimum value of $\|\mathsf{OP}(\boldsymbol{v}'' - \mathsf{OP}(\boldsymbol{v}))\|_p$ among all the τ-grid points $\mathsf{OP}(\boldsymbol{v}')$ such that $\mathsf{OP}(\boldsymbol{v}') \in \mathsf{Ball}(\mathsf{OP}(\boldsymbol{v}), L^p, d)$ and their corresponding inputs are adversarial examples.

Finally, we notice that the minimum value of $\|\mathsf{OP}(\boldsymbol{v}') - \mathsf{OP}(\boldsymbol{v})\|_p$ is equivalent to the optical value required by Equation (6.2). □

6.3 Computation of the Converging Upper and Lower Bounds

Since we demonstrated in the complexity analysis (Theorem 10 of Chapter 5) that the exact value of optimal reward is computationally hard, we approximate the reward by a decreasing upper bound and an increasing lower bound - the former is achieved via a gradient-based search algorithm and, for the latter, the Admissible A* approach is reused.

6.3.1 Upper Bound: Gradient-Based Search

We utilise a gradient-based search algorithm to compute an upper bound of `FMSR`. Below we introduce the concept of *spatial features* extracted from individual frames, and then construct the objective function and present the methods to calculate the gradients. As we utilise CNNs to extract the spatial features, and RNNs to capture the temporal dynamics between the neighbouring frames, we use $\mathcal{N}_{\mathsf{C}}$ to indicate the convolutional part of $\mathcal{N}$, and $\mathcal{N}_{\mathsf{R}}$ for the recurrent part.

Definition 6.8 (Spatial Features)**.** *Given a network $\mathcal{N}$, let $\mathcal{N}_{\mathsf{C}}$ denote the convolutional part, then $\mathcal{N}_{\mathsf{C}} : \boldsymbol{v} \to \eta \in \mathbb{R}^{l \times s}$ maps from input $\boldsymbol{v}$ to its extracted* spatial features *η, which has consistent length l of $\boldsymbol{v}$ and feature dimension* *s of a frame. Then, we pass η into the recurrent part $\mathcal{N}_{\mathsf{R}}$ and obtain the classification results, i.e., $\mathcal{N}_{\mathsf{R}} : \eta \rightarrow \mathcal{N}(\boldsymbol{v}, c), c \in C$.*

The objective is to manipulate optical flow as imperceptibly as possible while altering the final classification. We write the objective function as follows:

$$\forall t \in [1, l-1], t \in \mathbb{N}^{+},\ \min \mathtt{op}_t + \epsilon \cdot \boldsymbol{\nabla}_{\mathtt{op}_t}(\mathcal{N}, \boldsymbol{v})$$
$$s.t. \quad \boldsymbol{\nabla}_{\mathtt{op}_t}(\mathcal{N}, \boldsymbol{v}) = \frac{\partial \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\partial \eta} \odot \frac{\partial \eta}{\partial \mathtt{op}_t} \quad (6.14)$$

where ϵ is a constant, and $\boldsymbol{\nabla}_{\mathtt{op}_t}(\mathcal{N}, \boldsymbol{v})$ is the perturbation imposed on $\mathtt{op}_t$. The key point is to minimise $\boldsymbol{\nabla}_{\mathtt{op}_t}(\mathcal{N}, \boldsymbol{v})$ so that the perturbation is unnoticeable while simultaneously changing $\mathcal{N}(\boldsymbol{v})$. Here, we utilise the *loss* of $\mathcal{N}$ on $\boldsymbol{v}$, denoted as $\mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}$, to quantify the classification change. Intuitively, if $\mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}$ increases, $\mathcal{N}(\boldsymbol{v})$ is more likely to change. By utilising the concept of spatial features η, we rewrite $\boldsymbol{\nabla}_{\mathtt{op}_t}(\mathcal{N}, \boldsymbol{v})$ as $\frac{\partial \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\partial \eta} \odot \frac{\partial \eta}{\partial \mathtt{op}_t}$, where $\frac{\partial \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\partial \eta}$ denotes the gradient of the network's loss w.r.t the spatial features, $\frac{\partial \eta}{\partial \mathtt{op}_t}$ denotes the gradient of the spatial features w.r.t the optical flow, and $\odot$ denotes Hadamard/element-wise product. We introduce the computation of the two parts below.

Gradient of spatial features w.r.t. optical flow Here, $\frac{\partial \eta}{\partial \mathtt{op}_t}$ essentially exhibits the relation between spatial features and optical flow. Here we reuse input manipulation (Definition 6.2) to compute $\frac{\partial \eta}{\partial \mathtt{op}_t}$, though instead of manipulating the flow we impose perturbation directly on the frame. Intuitively, we manipulate the pixels of each frame to see how the subtle optical flow between the original and the manipulated frames will influence the spatial features. Each time we manipulate a single pixel of a frame, we get a new frame which is slightly different. If we perform $\delta_{\psi,\tau}$ on pixel $\mathcal{F}[m, n]$, and denote the manipulated frame as $\mathcal{F}_{m,n}$, its spatial features as $\eta_{m,n}$, the subtle optical flow between $\mathcal{F}_{m,n}$ and $\mathcal{F}$ as $\delta\mathtt{op}_{m,n}$, then $\frac{\partial \eta}{\partial \mathtt{op}_t}$ can be computed as in Equation (6.15).

$$\frac{\partial \eta}{\partial \mathtt{op}_t} = \begin{pmatrix} \frac{\|\eta_{1,1} - \eta\|_p}{\left\|\delta \mathtt{op}_{1,1}\right\|_p} & \cdots & \frac{\|\eta_{1,w} - \eta\|_p}{\left\|\delta \mathtt{op}_{1,w}\right\|_p} \\ \vdots & \ddots & \vdots \\ \frac{\|\eta_{h,1} - \eta\|_p}{\left\|\delta \mathtt{op}_{h,1}\right\|_p} & \cdots & \frac{\|\eta_{w,h} - \eta\|_p}{\left\|\delta \mathtt{op}_{w,h}\right\|_p} \end{pmatrix}_{w \times h} \tag{6.15}$$

Gradient of loss w.r.t. spatial features Meanwhile, $\frac{\partial \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\partial \eta}$ shows how the spatial features will influence the classification, which can be reflected by the loss of the network. After getting η from $\mathcal{N}_{\mathsf{C}}$, we can obtain $\mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}$ from $\mathcal{N}_{\mathsf{R}}$. If we perform pixel manipulation $\delta_{\psi,\tau}(\mathcal{F}[m,n])$ on frame $\mathcal{F}$, and obtain a new input, denoted as $\boldsymbol{v}_{\mathcal{F}[m,n]}$, then for this frame we have the gradient in Equation (6.16).

$$\frac{\partial \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\partial \eta} = \begin{pmatrix} \frac{\mathtt{loss}_{\boldsymbol{v}_{\mathcal{F}[1,1]}}^{\mathcal{N}} - \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\|\eta_{1,1} - \eta\|_p} & \cdots & \frac{\mathtt{loss}_{\boldsymbol{v}_{\mathcal{F}[1,w]}}^{\mathcal{N}} - \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\|\eta_{1,w} - \eta\|_p} \\ \vdots & \ddots & \vdots \\ \frac{\mathtt{loss}_{\boldsymbol{v}_{\mathcal{F}[h,1]}}^{\mathcal{N}} - \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\|\eta_{h,1} - \eta\|_p} & \cdots & \frac{\mathtt{loss}_{\boldsymbol{v}_{\mathcal{F}[w,h]}}^{\mathcal{N}} - \mathtt{loss}_{\boldsymbol{v}}^{\mathcal{N}}}{\|\eta_{w,h} - \eta\|_p} \end{pmatrix}_{w \times h} \tag{6.16}$$

Remark 4. *From Definition 6.8 of spatial features, i.e., $\eta = \mathcal{N}_{\mathsf{C}}(\boldsymbol{v})$, we know that the spatial features η only depend on each individual $\mathcal{F}$ of $\boldsymbol{v}$ and do not capture the temporal information between frames. That is, when $\mathcal{N}_{\mathsf{C}}$ remains unchanged, η and $\mathcal{F}$ have a direct relation, which indicates that the gradient of the latter can reflect that of the former. Therefore, during implementation, instead of the distance between $\eta_{m,n}$ and η, we calculate that between $\mathcal{F}_{m,n}$ and $\mathcal{F}$, i.e., $\|\mathcal{F}_{m,n} - \mathcal{F}\|_p$.*

Note that both gradients in Equations (6.15) and (6.16) have size $w \times h$, which is consistent with the size of an individual frame. After the $\odot$ operation of the Hadamard product, $\boldsymbol{\nabla}_{\mathtt{op}_t}(\mathcal{N}, \boldsymbol{v})$ essentially computes the perturbation on a single optical flow $\mathtt{op}_t, t \in [1, l-1]$. Therefore, for an input $\boldsymbol{v}$, we compute a series

Algorithm 4: Admissible A* for DNN Verification

```
Input  : A game model $\mathcal{G}(\mathcal{N}, \boldsymbol{v}, L^p, d)$, a terminating condition $tc$
Output : Lower bound of FMSR
1  procedure AdmissibleA*($\mathcal{G}(\mathcal{N}, \boldsymbol{v}, L^p, d), tc$):
2      $root \leftarrow s_0$ ;
3      while ($\neg tc$) do
4          $\mathsf{OP}(root) \leftarrow$ Player I($root$, optical flow = Farneback) ;
5          for $\mathsf{op}_t$ in $\mathsf{OP}(root)$ do
6              $\mathsf{op}_t[i] \leftarrow$ Player II($\mathsf{op}_t$) ;
7              $newnodes \leftarrow \delta_{\theta,\tau}(\mathsf{op}_t)(i)$ ;
8              for $node$ in $newnodes$ do
9                  $distances \leftarrow \mathsf{DistanceEstimation}(node)$ ;
10         $root \leftarrow \mathsf{MaximumSafeRadius}(distances)$ ;
11     return $\|\mathsf{OP}(root) - \mathsf{OP}(s_0)\|_p$
```

of perturbations on each optical flow generated from neighbouring frames, which is reflected in Equation (6.14).

6.3.2 Lower Bound: Admissible A*

We exploit admissible A* to compute the lower bound of Player I's reward, i.e., FMSR. An A* algorithm gradually unfolds the game model into a tree, in the sense that it maintains a set of children nodes of the expanded partial tree, and computes an estimate for each node. The key point is that in each iteration it selects the node with the *least* estimated value to expand. The estimation comprises two components: (1) the exact reward up to the current node, and (2) the estimated reward to reach the goal node. To guarantee the lower bound, we need to make sure that the estimated reward is minimal. For this part, we let the A* algorithm be *admissible*, which means that, given a current node, it never overestimates the reward to the terminal goal state. For each state s in the game model $\mathcal{G}$, we assign an estimated distance value

$$\mathsf{DistanceEstimation}(s) = \|\mathsf{OP}(s) - \mathsf{OP}(s_0)\|_p + heuristic(\mathsf{OP}(s)), \tag{6.17}$$

where $\|\mathsf{OP}(s) - \mathsf{OP}(s_0)\|_p$ is the distance from the original state s_0 to the current state s based on the L^p norm, and $heuristic(\mathsf{OP}(s))$ is the admissible heuristic function that estimates the distance from the current state s to the terminal state. Here, we use $\tilde{d}(L^p, \tau)$ in Equation (6.4).

We present the admissible A* algorithm in Algorithm 4. To be more specific, in each iteration for the *root* node (initialised as the original input s_0), Player I selects among a series of optical flows $\mathsf{OP}(root)$ extracted by an optical flow method such as the Gunnar Farnebäck algorithm [17]. We remark that various optical flow methods, e.g., those mentioned in Section 3.4.3, can fit into our framework as long as a dense flow is produced. Subsequently, on each chosen optical flow op_t Player II determines the dimensions $\mathsf{op}_t[i]$ to be perturbed (Lines 1-6). On each of the dimensions, the atomic manipulation function applies $\delta_{\theta,\tau}$, which essentially places adversarial perturbations on the optical flow, and after applying the manipulated flows onto the current *root*, a set of manipulated inputs become the *newnodes* (Line 7). Then, the DistanceEstimation function in Equation (6.17) computes the distances of the *newnodes* to the original optical flow set $\mathsf{OP}(\boldsymbol{v})$ and puts them into the *distances* set, which maintains the estimated distance values for all the leaf nodes (Lines 8-11). Eventually, among all the leaf nodes in *distances*, the MaximumSafeRadius function sets the one with the *minimal* distance as the new *root* (Line 12). Finally, at the end of each iteration, the terminating condition *tc* is checked: if it is not satisfied, a new iteration starts; otherwise, the optimal distance value $\|\mathsf{OP}(root) - \mathsf{OP}(s_0)\|_p$, i.e., lower bound of FMSR, is returned.

6.4 Experimental Results

This section presents the evaluation results of our framework regarding the robustness guarantees for the deep neural networks on the video dataset *UCF101* [65]. Specifically, we develop the tool DeepVideo and perform the experiments on a Linux server with the NVIDIA GeForce GTX Titan Black GPUs and the Ubuntu 14.04.3 LTS operating system.

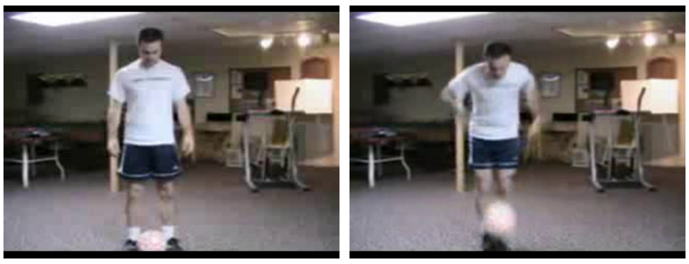
6.4.1 Network Architecture

In the experiments, we exploit a VGG16 [64] + LSTM [25] architecture, in the sense of utilising the *VGG16* network to extract the spatial features from the UCF101 video dataset and then passing these features to a separate RNN unit *LSTM*. For each video, we sample a frame every 1000 ms and stitch them together into a sequence of frames. Specifically, we run every frame from every video through VGG16 with input size $224 \times 224 \times 3$, excluding the top classification part of the network, i.e., saving the output from the final Max-Pooling layer. Hence, for each video, we retrieve a sequence of extracted spatial features. Subsequently, we pass the features into a single LSTM layer, followed by a Dense layer with some Dropout in between. Eventually, after the final Dense layer with activation function Softmax, we get the classification outcome.

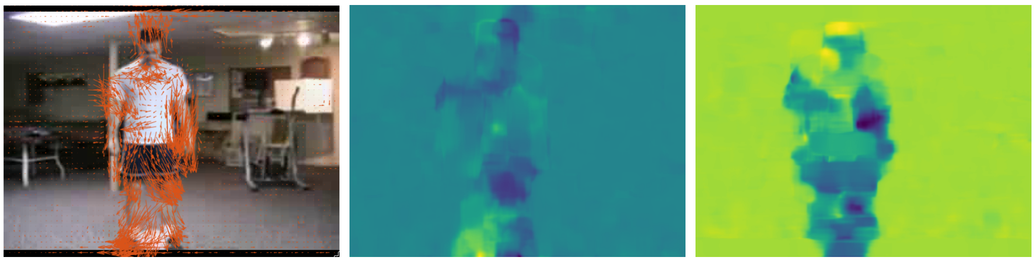
We use the categorical cross-entropy loss function and the accuracy metrics for both the VGG16 and LSTM models. Whilst the former has a SGD optimiser and directly exploits the imagenet weights, we train the latter through a rmsprop optimiser and get 99.15% training accuracy as well as 99.72% testing accuracy. Specifically, when the loss difference cannot reflect the subtle perturbation on optical flow during the computation of upper bounds, we use the discrepancy of logit values instead.

6.4.2 Adversarial Examples via Manipulating Optical Flows

We illustrate how *optical flow* can capture the temporal dynamics of the moving objects in neighbouring frames. In this case, we exploit the *Gunnar Farnebäck* algorithm [17] as it computes the optical flow for all the pixels in a frame, i.e., *dense* optical flow, instead of a sparse feature set. We remark that the proposed framework can work with *any* state-of-the-art optical flow algorithm as long as a dense flow is produced – here we chose the Farnebäck approach because it is easily accessible in Python and Matlab. In Figure 6.10 (top) we also exploited another *Horn-Schunck*



(a) Soccer Juggling at 0 s and 1 s.



(b) Optical flow (red arrows) and its magnitude (left) and direction (right).

Figure 6.3: Illustration of how an optical flow can capture the dynamics of the moving objects between two adjacent frames sampled from a video. (a): Two sampled frames from Soccer Juggling with original size $320 \times 240 \times 3$. (b): The optical flow (red arrows) extracted between the frames, and its two characteristics: magnitude and direction.

flow method. Figure 6.3 presents an optical flow generated from two adjacent frames of a video labelled as Soccer Juggling: (a) shows two frames sampled at 0 s and 1 s of the video; and (b) exhibits the characteristics of the flow: magnitude and direction. We observe that, while the indoor background essentially remains unchanged, the motion of the player together with the football is clearly captured by the flow.

Moreover, we include an example of the *sequential* optical flows extracted from another video with classification Balance Beam in Figure 6.4, where the top row exhibits four sampled frames from 0 s to 3 s, extracted from which the corresponding optical flows (in blue arrows) are shown in the 2nd row, with two characteristics magnitude and direction in the 3rd and the bottom rows, respectively. Overall, the motions of the gymnast performing on the balance beam in the centre of the video are captured in the highlighted regions of the magnitude and direction characteristics.

We now demonstrate how a very slight *perturbation on the flow*, almost imperceptible to human eyes, can lead to a misclassification of the whole video. Figure 6.5

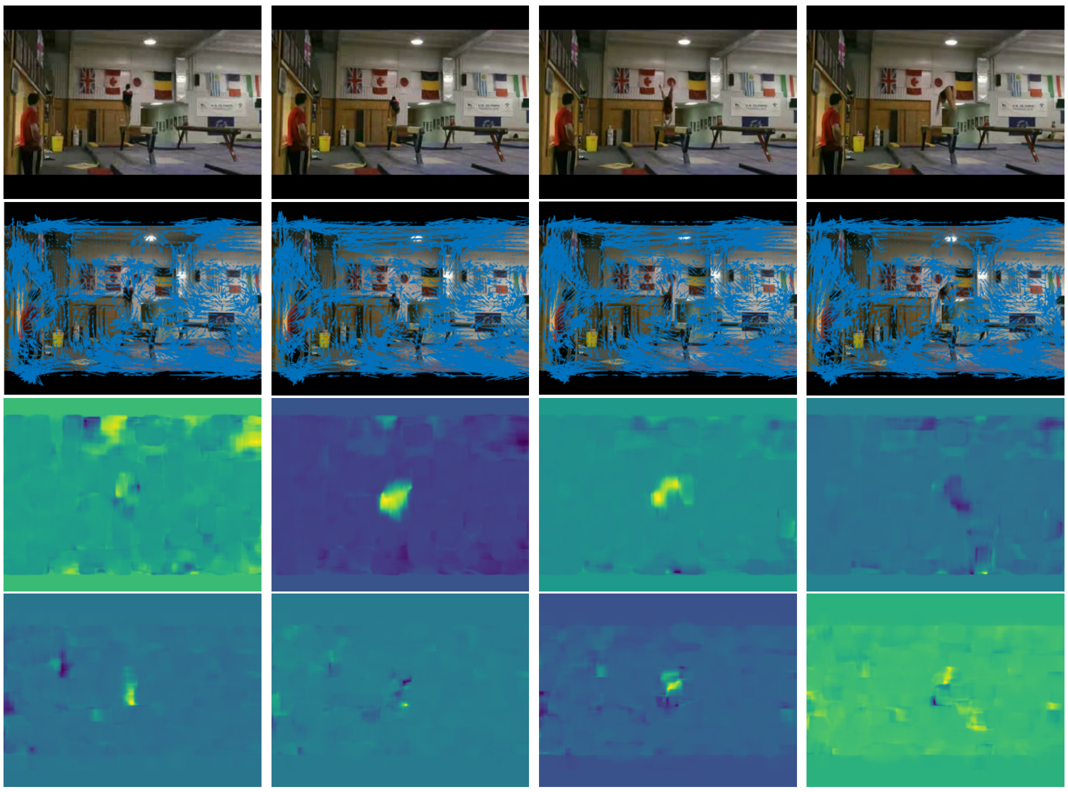


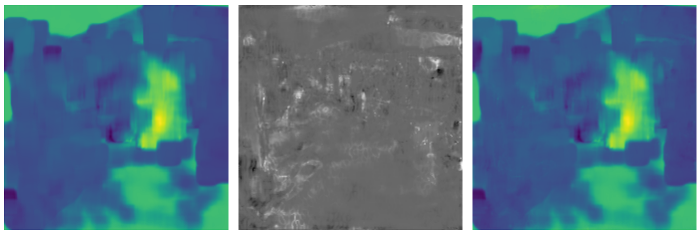
Figure 6.4: Examples of a sequence of optical flows extracted from a Balance Beam video. Top row: four sampled frames from 0 s to 3 s with original size $320 \times 240 \times 3$. 2nd row: the optical flows (blue arrows) extracted between the frames. 3rd row: one of optical flow's characteristics: magnitude. Bottom row: the direction of optical flow.

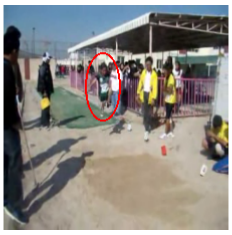
exhibits that a video originally classified as Long Jump with confidence 100.00% is manipulated into Floor Gymnastics with confidence 86.10%. Two sampled frames at 1 s and 2 s are shown in the 1st column. If we compare the original optical flow of magnitude and direction (2nd column) generated from the frames with the perturbed ones (4th column), we can hardly notice the difference (3rd column). However, the classification of the video has changed.

6.4.3 Converging Upper and Lower Bounds

We illustrate the convergence of the bound computation for the *maximum safe radius* with respect to manipulations on the optical flows extracted from the consecutive frames of a video. Take a Hammer Throw video as an example. Figure 6.6 exhibits four sampled frames (top row) from a Hammer Throw video and the optical flows

(a) Long Jump at 1 s.


(b) Optical flow magnitude: original, difference, and perturbed.


(c) Long Jump at 2 s.

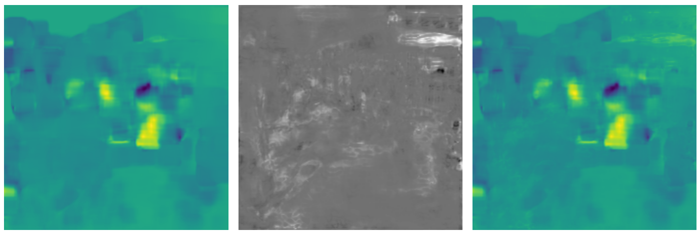

(d) Optical flow direction: original, difference, and perturbed.

Figure 6.5: Imperceptible perturbations on optical flow, in terms of magnitude and direction, leading to misclassification from Long Jump (100.00%) to Floor Gymnastics (86.10%). (a)(c): sampled frames at 1 s and 2 s with size $224 \times 224 \times 3$. (b): original and perturbed magnitude. (d): original and perturbed direction.

extracted between them (2nd row). The descending upper bounds (yellow) and the ascending lower bounds (blue) to approximate the value of MSR are presented in Figure 6.7(a). Intuitively, after 20 iterations of the gradient-based algorithm, the upper bound, i.e., minimum distance to an adversarial example, is 5670.31 based on the L^2 distance metric. That is, manipulations imposed on the flows exceeding this upper bound may be *unsafe*. Figure 6.6 (3rd row) shows some of such unsafe perturbations on each optical flow, which result in the misclassification of the video into Front Crawl with confidence 99.86%. As for the lower bound, we observe that, after 1000 iterations of the admissible A* algorithm, the lower bound reaches 52.95. That is, manipulations within this L^2 norm ball is absolutely *safe*. Some of such safe perturbations can be found in the bottom row of Figure 6.6.

We include another example to illustrate the convergence of the upper and lower bounds. Similarly, Figure 6.8 exhibits five sampled frames (top row) and the optical flows extracted between them (2nd row). By utilising our framework, we present the approximation of MSR in Figure 6.7(b), where the yellow line indicates the

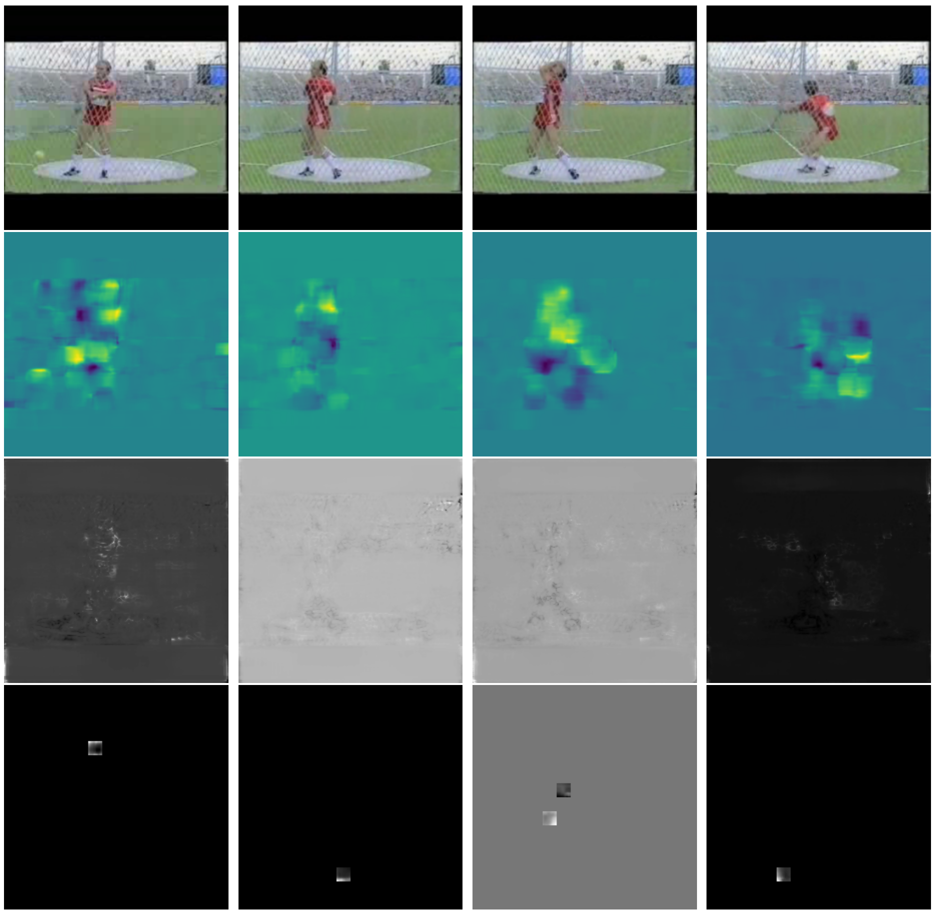


Figure 6.6: Examples of *unsafe* and *safe* perturbations on the optical flows of a Hammer Throw video. Top row: four sampled frames from 0 s to 3 s. 2nd row: optical flows of the frames from 0 s to 3 s. 3rd row: *unsafe* perturbations on the flows corresponding to the upper bound. Bottom row: *safe* perturbations corresponding to the lower bound.

descending trend of the *upper* bound, whereas the blue line denotes the ascending trend of the *lower* bound. Intuitively, after 20 iterations of the gradient-based algorithm, the upper bound, i.e., minimum distance to an adversarial example, is 2100.45 based on the L^2 distance metric. That is, manipulations imposed on the flows exceeding this upper bound may be *unsafe*. Figure 6.8 (3rd row) shows some of such unsafe perturbations on each optical flow, which result in the misclassification of the video into Front Crawl with confidence 97.04%. As for the lower bound, we observe that, after 1500 iterations of the admissible A* algorithm, the lower bound reaches 146.61. That is, manipulations within this L^2 norm ball is absolutely *safe*.

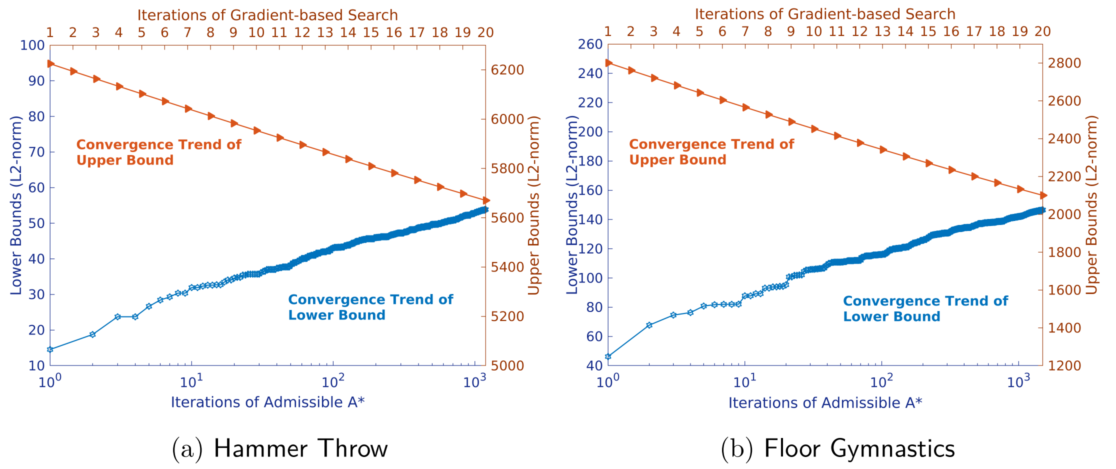


Figure 6.7: Convergence of the *maximum safe radius* with respect to manipulations on extracted optical flows. The blue line denotes the decreasing *upper* bound from the gradient-based algorithm, and the black line denotes the increasing *lower* bound from admissible A*. (a) The Floor Gymnastics video. (b) The Hammer Throw video.

Some of such safe perturbations can be found in the bottom row of Figure 6.8.

6.4.4 Extension to Naturally Plausible Distortions

Apart from the above-mentioned adversarial perturbations imposed on the optical flows, no matter whether safe or unsafe, our framework can be extended to distortions that are more *natural and physically plausible* to the modality of the data itself. This is because all the large perturbations that preserve the semantic content of a video, such as "brightness change", "camera occlusion", "horizontal flip", and "angular rotation", are essentially compositions of various atomic manipulations on the videos, and thus can easily be incorporated.

Take the "*brightness change*" perturbation as an example. As illustrated in Figure 6.9, we increase the brightness of the Hammer Throw video on the frame level. That is, each pixel in the same frame is simultaneously brightened by the atomic manipulation τ, thus resulting in the overall distance to the original video increasing by $\tilde{d}'(L^p, \tau \cdot w \cdot h)$, where w denotes the width of the frame and h height. The corresponding lower bounds of `MSR` are computed in Figure 6.9(c). Intuitively, it means that any degree of brightness alteration is definitely *safe* as long as the

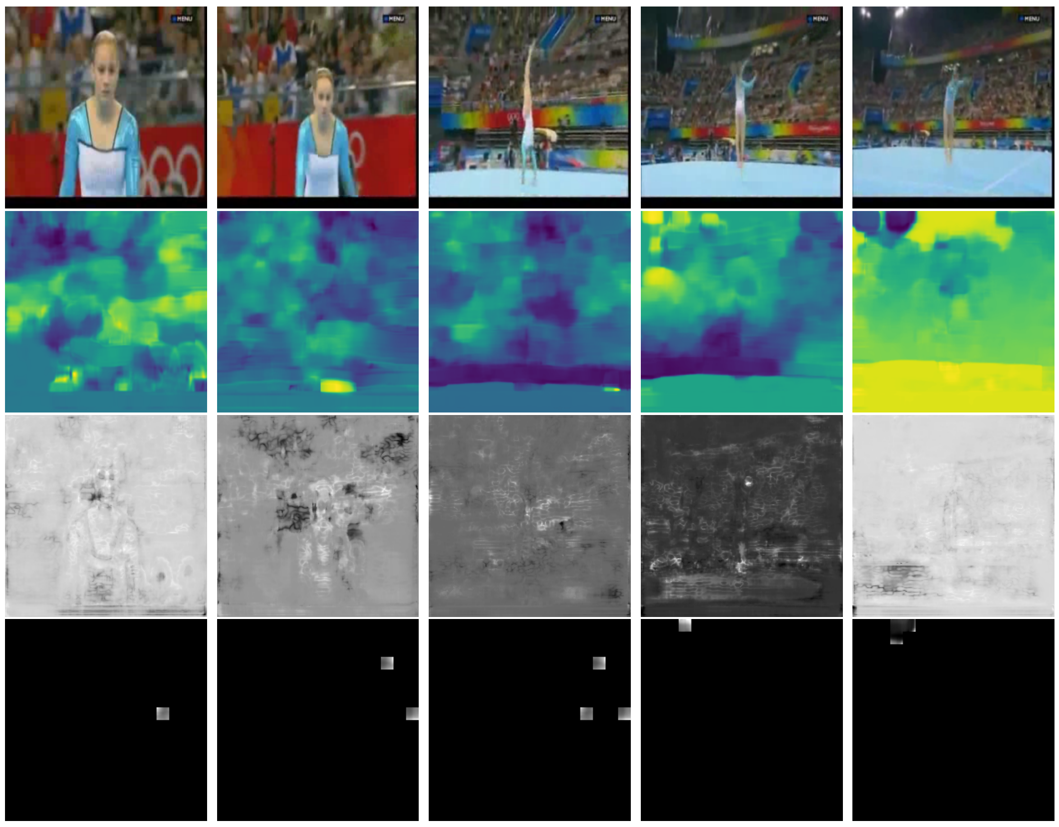
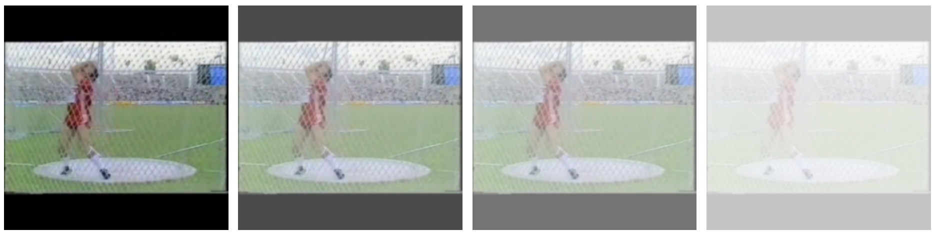
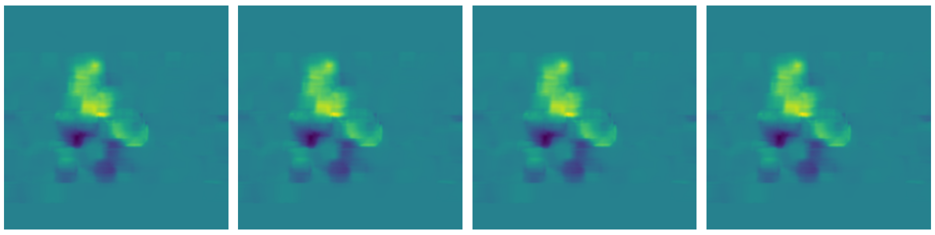


Figure 6.8: Examples of *unsafe* and *safe* perturbations on the optical flows of a Floor Gymnastics video. Top row: five sampled frames from 0 s to 4 s. 2nd row: optical flows of the frames from 0 s to 5 s. 3rd row: *unsafe* perturbations on the flows corresponding to the upper bound. Bottom row: *safe* perturbations corresponding to the lower bound.

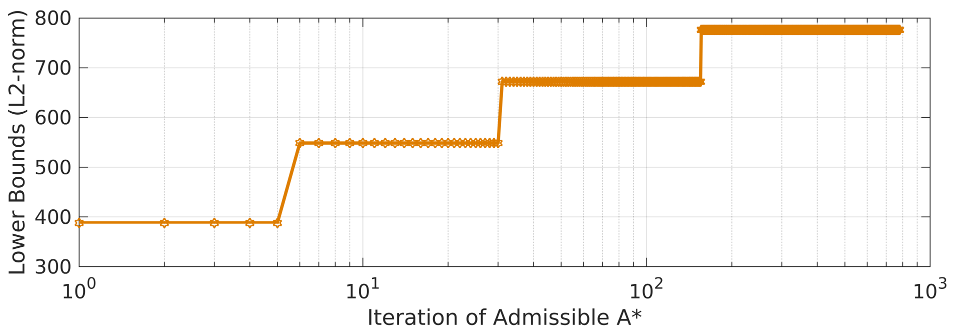
distance to the original video is less than the computed lower bound. For instance, after 10 iterations, the lower bound is 548.68 based on the L^2 norm, then any frame-level brightness increase less than 548.68 in the Euclidean distance will not change the classification of this video. Besides, compared to the bounds produced in Figure 6.7, here the ascending trend of the lower bounds is slightly different, in the sense that, every once in a while, the bounds remain unchanged for a number of iterations, e.g., same 387.97 during the first 5 iterations, corresponding to the total 5 frames this video contains. This is because the brightness perturbation is imposed at the frame level, and in each iteration when Player I chooses a frame to perturb, Player II adds the same $\tau \cdot w \cdot h$ to this frame. In other words, Player I has to traverse every frame (and their combinations after the first 5 iterations)



(a) Brightness increase of the Hammer Throw video.



(b) Optical flows corresponding to the same frame with brightness changes.



(c) Lower bounds of the maximum safe radius.

Figure 6.9: Safe *brightness* changes to the Hammer Throw video and the corresponding lower bounds of the maximum safe radius. (a) The frame at 2 s of Hammer Throw with increasing brightness. (b) The optical flows extracted from the same frame taking into account the brightness change. (c) The ascending lower bounds of the maximum safe radius reflecting the brightness change.

to increase the lower bounds.

One interesting phenomenon observed is that, as exhibited in Figure 6.9(b), when the brightness of a frame increases, the extracted optical flow on the same frame is *not* significantly affected, due to the fact that the motion is relatively unchanged. In other words, optical flow can naturally discard some perturbations that do not alter the underlying temporal dynamics.

Apart from the "brightness change", we include some other possible natural

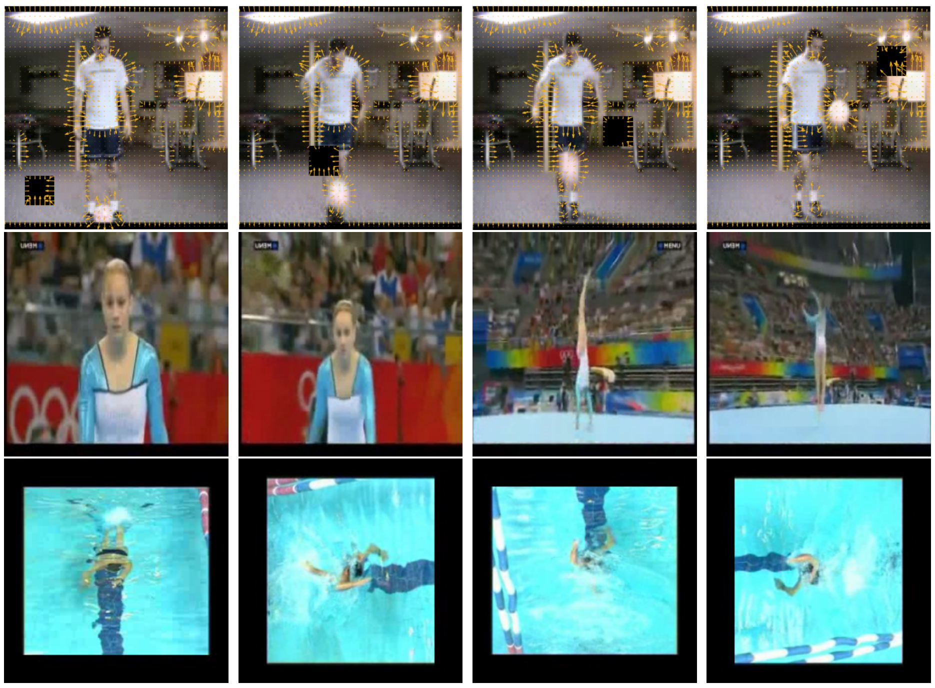


Figure 6.10: Some possible extensions of the adversarial perturbations to more *naturally plausible distortions* such as "camera occlusion", "horizontal flip", and "angular rotation". Top: "camera occlusion" to the Soccer Juggling video with the *Horn-Schunck* [26] optical flow method. Middle: "left and right flip" to the Floor Gymnastics video. Bottom: "angular rotation" to the Front Crawl video.

distortions of the adversarial perturbations in Figure 6.10, such as the "camera occlusion" of the Soccer Juggling video with the *Horn-Schunck* optical flow method, the "horizontal flip" of the Floor Gymnastics video, and the "angular rotation" by 90°, 180°, and 270° of the Front Crawl video. We can see that the "camera occlusion" here is very similar to the safe perturbations of the Hammer Throw video in Figure 6.6 (bottom row), and thus can be handled using similar methods. The "horizontal flip" and the "angular rotation" involve manipulations that form a group, and to deal with those our approach would need to be extended, for example by incorporating network invariances. This is left as future work.

Finally, regarding these various adversarial perturbations, we remark that whether the perturbations are visible for a human largely depends on the *manipulation* manner – with the same distance to the original input, manipulations such as these physically plausible distortions are certainly more visible than

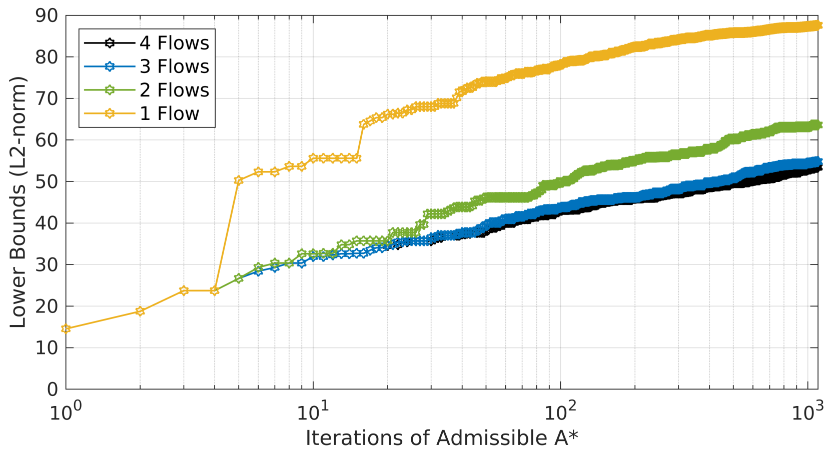


Figure 6.11: Scalability of the *maximum safe radius* with respect to optical flow. Lines with different colours represent the lower bounds of the *maximum safe radius* of a Hammer Throw video with different dimensions of the manipulated optical flows.

unsafe perturbations produced by the gradient-based search algorithm and the safe perturbations created from the admissible A* algorithm.

6.4.5 Efficiency and Scalability

As for the *computation time*, the upper bound requires the gradient of optical flow with respect to the frame, and because we extract dense optical flow, the algorithm needs to traverse each pixel of a frame to impose atomic manipulations; thus it takes around 30 minutes to retrieve the gradient of each frame. Once the gradient of the whole video is obtained, and the framework enters into the cooperative game, i.e., the expansion of the tree, each iteration takes minutes. Meanwhile, for the lower bound, the admissible A* algorithm expands the game tree in each iteration which takes minutes, and updates the lower bound wherever applicable. Note that initially the lower bound may be updated in each iteration, but when the size of the game tree increases, it can take hours to update.

Moreover, we analyse the *scalability* of our framework via an example of a Hammer Throw video in Figure 6.11, which shows the lower bounds of the maximum safe radius obtained with respect to different dimensions of the manipulated optical flows. We observe that, within the same number of iterations, smaller input dimension leads to faster convergence.

6.5 Summary

In this chapter, we study the *maximum safe radius* problem of neural networks, including CNNs and RNNs, with respect to the optical flow sets extracted from sequential videos. By relying on Lipschitz continuity, we transform the problem to a finite optimisation whose approximation has provable guarantees, and subsequently reduce the finite optimisation to the solution of a two-player turn-based game. We design algorithms to compute upper and lower bounds, and demonstrate the convergence trend of the bounds in the experiments.

7 Conclusions

7.1 Summary

The central question examined in this thesis is how to make sure deep neural networks behave correctly in real-world safety-critical scenarios, such as autonomous driving and automatic medical diagnosis, where erroneous behaviours might cause tremendous loss. A particular category of erroneous behaviours that have safety implications is the vulnerability of neural networks to *adversarial examples*. Researchers found that, even a well-trained network performing at a human level accuracy might, surprisingly, have a 100% error rate on inputs that are intentionally constructed.

Regarding this, this thesis studies the *robustness* property of neural networks, defined as the invariance of a network's decision-making procedure against small perturbations in the neighbourhood of an input. Early endeavours to evaluate robustness emerged in the computer vision and security communities, where various attack techniques have been developed to craft such adversarial examples on purpose. And whether a network can be easily attacked is used as an indication of the network' (lack of) robustness. However, such assessment does not have *guarantees*, in the sense that it cannot tell whether a misclassification is definitely not going to happen, given some constraints such as the size of the neighbourhood. Later, around 2016, researchers in the field of formal verification started to propose approaches that can provide guarantees. For example, [30] computes deterministic guarantees on small size networks, and [28] deploys a layer-by-layer exhaustive search.

Compared with existing works in literature, the methodologies developed in this thesis for the *robustness evaluation of deep neural networks with provable guarantees* are scalable to large neural networks and applicable to real-world systems. Specifically, we start with the assessment of local robustness on pixel-level images, as presented in Chapter 4. Here, to measure the discrepancy between an original input and the manipulated one, we exploit the Hamming distance, which is an "interesting" yet "challenging" metric – "interesting" in the sense that it can intuitively and straightforwardly reflect the perturbations imposed on an image, e.g., a tiny mud speckle on a traffic sign covering ten pixels; "challenging" in the sense that it is non-differentiable, unlike the L^p norms, and therefore most optimisation algorithms utilising gradient descent do not work in this case. Moreover, in this chapter, we extend the local robustness property to the global robustness so that we enable the evaluation of a network on an entire dataset instead of just a single image. To the best of our knowledge, this is the *first* algorithm that evaluates global robustness of networks with provable guarantees based on the Hamming distance. Apart from the pixel-level robustness evaluation, in Chapter 5, we extend the robustness of neural networks to feature level, which makes more sense because, when humans perceive an image, we immediately identify the salient features and almost instantly exclude the irrelevant elements out of subconsciousness, e.g., pixels in the background. Similarly, as far as we know, it is the *first* work that guarantees feature-level robustness of neural networks. To take a step further, in Chapter 6, we not only consider the spatial features in images, as in the above two chapters, where convolutional networks suffice, but also take into account the temporal dynamics between adjacent frames from a video, for which recurrent networks are deployed. By the time this work was put online, there was *no other* work in public domain addressing the robustness guarantees for videos.

Contributions

We summarise the main contributions of this thesis in the following aspects:

- To quantify robustness, we exploit the concept called the *maximum safe radius* of a neural network with respect to an input, which is defined as the distance, measured by a certain metric, to the original input in the sense that, if exceeding the distance, there definitely *exists* an adversarial example in the input space, whereas, within the distance, *all* the input points are safe. Specifically, in Chapter 4 we compute this maximum safe radius when the input is an image and the metric is the Hamming distance, and subsequently extend to other metrics such as the Manhattan and the Euclidean distances, i.e., the L_1 and L_2 norms, in Chapter 5. Moreover, in Chapter 6 we extend the input space from images to videos, and compute the *maximum safe radius with respect to optical flow*, so that, for a time-series of frames sampled from a video, both spatial features in individual frames and temporal dynamics between adjacent frames can be captured. While the maximum safe radius in these scenarios is regarded as *local robustness*, in Chapter 4 we also extend to *global robustness* as an expectation of the maximum safe radius over a dataset of independent and identically distributed images. Apart from this, to evaluate the robustness of neural networks on features of an image, in Chapter 5 we propose the *feature robustness* problem to find the most robust feature of an image by restricting adversarial perturbations to certain features, and thus the existence of adversarial attacks is controllable.

- To solve the above maximum safe radius and feature robustness problems, theoretically, we need to consider all the possible input points in the norm ball, of which, however, there are *infinitely* many. Regarding this, we utilise the fact that the neural networks studied in this thesis are *Lipschitz continuous* to discretise the neighbourhood space of an input, in other words, to transform the infinite number of points in the norm ball into a *finite* number of uniformly distributed points on a grid. We demonstrate that, when the distances between the grid points are small, we can reduce the robustness evaluation to finite optimisation problems that have *provable guarantees*. This is the underlying technique we use in Chapters 4, 5, and 6 to provide the robustness evaluation of deep neural networks with guarantees.

- Because our work is scalable to large-scale state-of-the-art neural networks, e.g., VGG16 and ResNet101, and applicable to real-world systems such as The German Traffic Sign Recognition Benchmark (GTSRB) dataset, it is not always realistic to compute the exact maximum safe radius or the exact feature robustness, as the input dimensions can be huge, and the networks usually contain millions of hidden neurons. To deal with this, we calculate two bounds, namely an *upper bound* and a *lower bound*, in the sense that the upper bound is monotonically decreasing whilst the lower bound monotonically increasing, so that eventually they will *converge* to the optimal value, i.e., the exact value of maximum safe radius or the feature robustness. Intuitively, it means that all the points with distance to the original input less than the lower bound are definitely safe, and when the distance to the original input exceeds the upper bound, there definitely exists some point that is an adversarial example. We deploy different methods to obtain the bounds, for example, in Chapter 4 the pixel-level subspace sensitivity is analysed and a generated *saliency map* is used to compute the bounds over the local/global maximum safe radius. For robustness on features and videos in Chapters 5 and 6, we solve the finite optimisation problems via a game-based verification framework, which is introduced below.

- Unlike the robustness evaluation of networks on pixel-level images, where adversarial perturbations may be applied to any arbitrary pixels, robustness evaluation on the features of an image or the videos needs to determine, within which specific feature, or which optical flow extracted between adjacent frames, modifications should occur. In these two scenarios, we utilise *a game-based verification framework* to solve the finite optimisation problems. Specifically, for a two-player turn-based game, in Chapter 5, we let Player I select features and Player II choose pixels/channels within the selected feature to perform manipulations, and in Chapter 6 we let Player I pick optical flows and Player II determine dimensions within the picked optical flow to conduct alterations. In each iteration of the game, after both players have made their choices, the input image/video is perturbed, and the game continues. For the robustness evaluation on features, we set Player II's aim as to minimise the distance to an adversarial example, then when Player I is *cooperative*, this is essentially the maximum safe radius computation problem, whereas when Player II is *competitive*, this addresses the feature robustness problem. We apply the Monte Carlo tree search algorithm to compute the upper bounds for both cooperative and competitive games, and admissible A* and Alpha-Beta Pruning, respectively, to compute the lower bounds for the two game-based approaches. Similarly, for video robustness, we exploit a gradient-based algorithm to improve the upper bounds and the Admissible A* to refine the lower bounds. In either case, we demonstrate that the solution to the optimisation problems is Player I's reward when taking the optimal strategy.

To add to the discussion, we also mention the key potential limitation of the methods proposed in this thesis. Our approach assumes the knowledge of a Lipschitz constant of the neural networks, though not necessarily a tight one. That is, the value of Lip should satisfy the Lipschitzian property (Definition 3.15). In the literature, there are several techniques to estimate such a constant, for example FastLin/FastLip [74], Crown [82], and DeepGO [59]. The size of the Lipschitz constant is inversely proportional to the grid width and thus the error bound, and therefore affects the computational performance. Due to the high non-linearity and high-dimensionality of modern deep neural networks, we remark that it is non-trivial to conduct verification even if the Lipschitz constant is known.

7.2 Future Work

Several directions for potential future research arise from the work in this thesis. Below we highlight several that are of particular relevance.

Classes of Adversarial Manipulations

To this point, most of our work has focused on adversarial perturbations accumulated from imposing *atomic* manipulations in input dimensions. Especially when generating a lower bound, we need to make sure all the inputs with distances less than the lower bound must be safe, therefore, in each iteration, the modification of the bound is caused by a very slight alteration constrained by the Lipschitz constant to provide provable guarantees. Nevertheless, such modifications often tend to be random and unrealistic to human perception. Researchers arguably even doubt if they are actually natural and physically plausible.

Regarding this, a possible future work is to consider adversarial perturbations that preserve the *semantic content* of the input data, for example, the "brightness change", "camera occlusion", "left/right flip", and "angular rotation" distortions for the robustness evaluation on videos, as studied in Chapter 6.

Moreover, we observe that such kind of physically plausible distortions often have their own characteristics, and thus can be categorised into specific *classes*. For instance, to measure the discrepancy caused by "brightness change", the Chebyshev distance might be one of the appropriate metrics, as the brightness in all the pixels usually change simultaneously by the same magnitude. Also, as demonstrated in our work, "brightness change" can easily be easily handled via optical flow, as when the brightness increases or decreases, the motion of the objects is virtually unaffected. As for "camera occlusion", one of its characteristics is that the shape and size of the occlusion often remain unchanged. In other words, the distance between the input and the perturbed image does not change much. Compared with those, "horizontal flip" and "angular rotation" are similar in the sense that it would seem slightly inappropriate to utilise the conventional L^p norms to measure the distance in a dimension-by-dimension manner. Instead, it is necessary to consider operations that distort or rotate images, and extend robustness evaluation to take this into account.

Robustness of Natural Language Processing

Another appealing future work is the possible application to natural language input, i.e., the *robustness evaluation of deep neural networks on natural language processing.* Neural networks have been widely employed in this domain, such as machine translation, language modelling, and sentiment analysis. For example, in a sentence-level sentiment analysis task, adversarial perturbations such as replacing the word "excellent" by "terrific" in a given text may completely change the classified polarity from positive to negative.

In fact, if we compare the similarity between the robustness evaluation of natural language processing and the work in this thesis, there is a high chance that our proposed methodologies can be adopted. To be more specific, if given a trained network $\mathcal{N}$ to perform polarity classification, and a text represented by word embedding with finite words $\boldsymbol{t} = \{w_1, \ldots, w_m\}$, $m \in \mathbb{N}^+$, we can define the *local robustness* of the network with respect to the text as the invariance of $\mathcal{N}$'s decision-making in the neighbourhood of $\boldsymbol{t}$. This neighbourhood can be expressed by our metric ball $\mathsf{Ball}(\boldsymbol{t}, \mathfrak{d}_{\mathsf{Embed}}, d)$, where $\mathfrak{d}_{\mathsf{Embed}}$ can be a similarity measurement from word embedding. Thus, the *verification of local robustness* problem is to determine whether $\textsc{Robust}(\mathcal{N}, \mathsf{Ball}(\boldsymbol{t}, \mathfrak{d}_{\mathsf{Embed}}, d)) = \texttt{True}$ holds, as in our Definition 3.9.

7.3 Outlook

This thesis has developed methodologies for the robustness evaluation of deep neural networks with provable guarantees. Compared to traditional formal verification disciplines, the safety verification of deep neural networks essentially emerged during my doctoral study, and thus is still in relative infancy. Nevertheless, with the world-wide application of deep learning techniques, we believe that methods to provide robustness guarantees for deep neural networks will constitute a key element of safety-critical real-world systems in future.

Appendices

A Experimental Settings for DeepTRE

A.1 ImageNet Saliency Maps and Local Robustness

A.1.1 State-of-the-Art ImageNet Models

- AlexNet [34] – a convolutional neural network with 8 layers, designed by Alex Krizhevsky. It competed in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [62], and achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up.
- VGG-16 and VGG-19 [64] – two 16-layer and 19-layer convolutional networks released in 2014 by K. Simonyan and A. Zisserman from the Visual Geometry Group (VGG) at the University of Oxford. This family of architectures achieved 7.4% to 7.3% top-5 error in the ILSVRC 2014.
- ResNet50 and ResNet101 [24] – deep residual networks with a depth of 50 and 101 layers respectively proposed by Kaiming He et al. at Microsoft Research Asia. An ensemble of these networks achieved 3.57% error on the ImageNet test set and won the 1st place on the ILSVRC 2015 classification task.

A.1.2 Hardware/Software Platforms and Parameter Settings

Hardware and Software Platforms

- Hardware Platform:
  - NVIDIA GeForce GTX 1050
  - Intel® Core™ i7-7700HQ Processor
- Software Platform:
  - MATLAB 2018a
  - Neural Network Toolbox
  - Image Processing Toolbox
  - Parallel Computing Toolbox
  - Pretrained ImageNet Models (AlexNet, VGG-16/19, ResNet50/101)

DeepTRE: Parameter Setting

- EPSILON = `0.3`
- Subspace_Dimension = `1`
- Tested Images: `20 ImageNet Images (Random)`

A.1.3 ImageNet Saliency Maps and Adversarial Images

Figures A.1, A.2, and A.3 exhibit more examples of adversarial images and saliency maps.

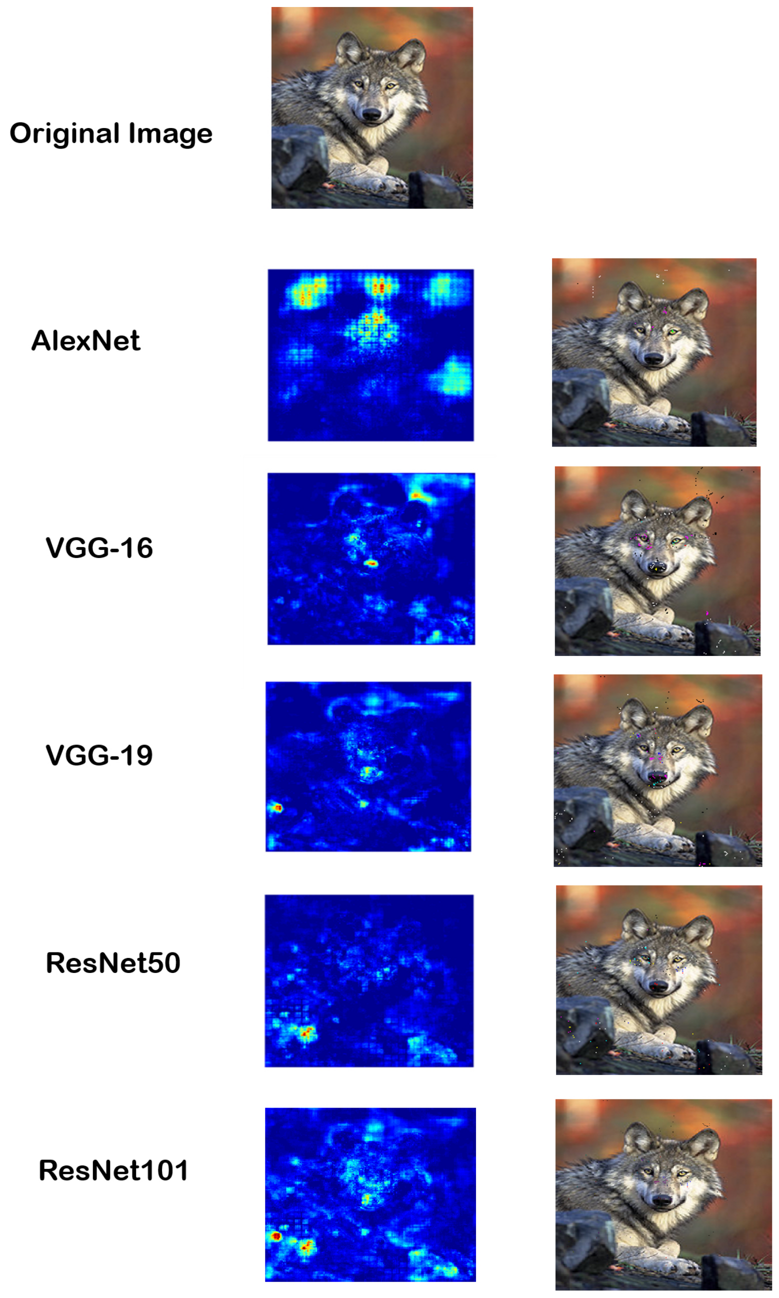


Figure A.1: Adversarial examples on upper boundaries returned by our tool DeepTRE (right column), and saliency maps for each ImageNet DNN model (left column).

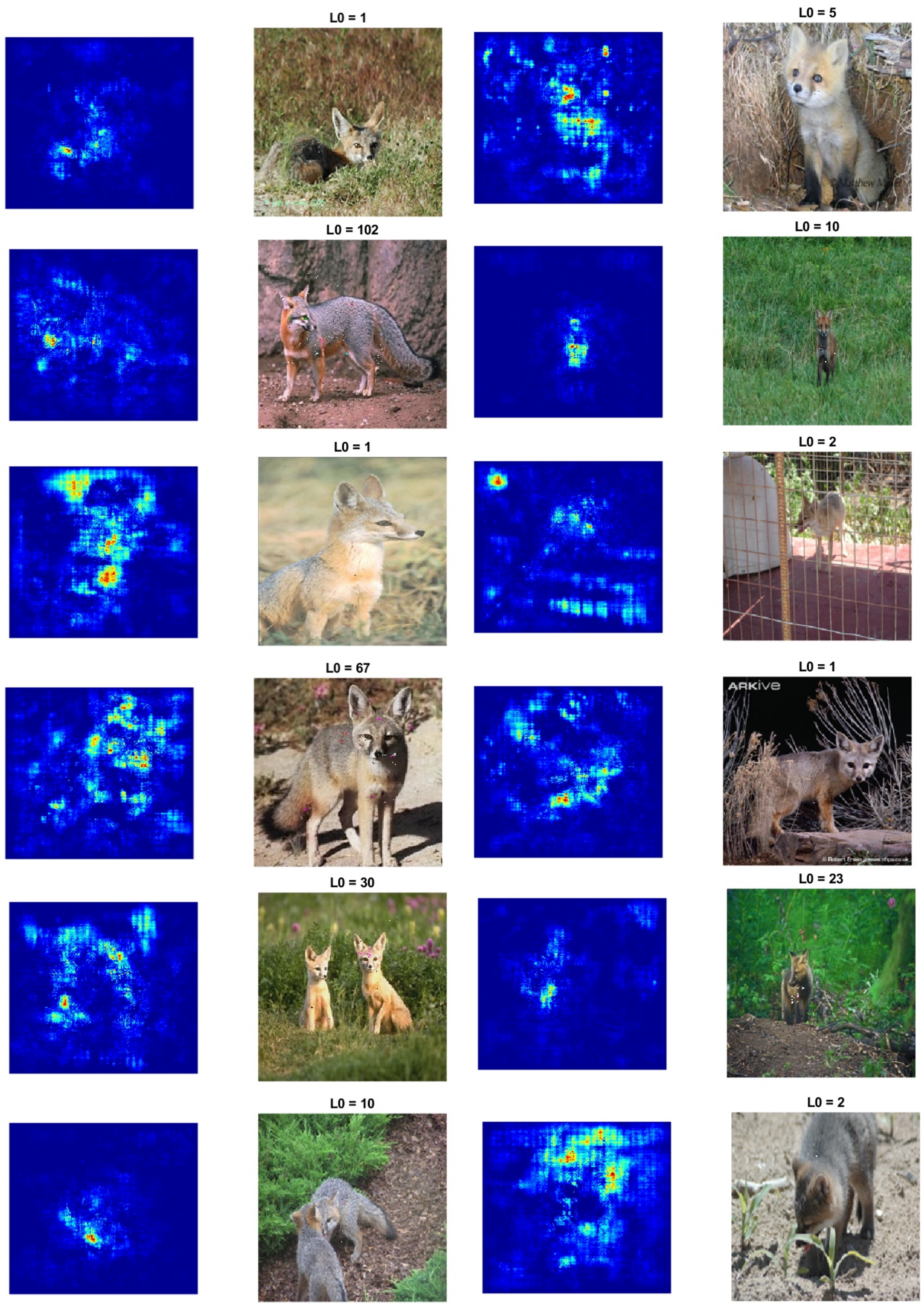


Figure A.2: Adversarial examples on upper boundaries (right column) returned by DeepTRE, and their saliency maps (left column) for the ImageNet AlexNet model. Note that all adversarial images with $\mathfrak{d}_{\mathsf{m}} = 1, 2$ are also the ground-truth Hamming distance adversarial images since their upper bounds and lower bounds local robustness have converged.

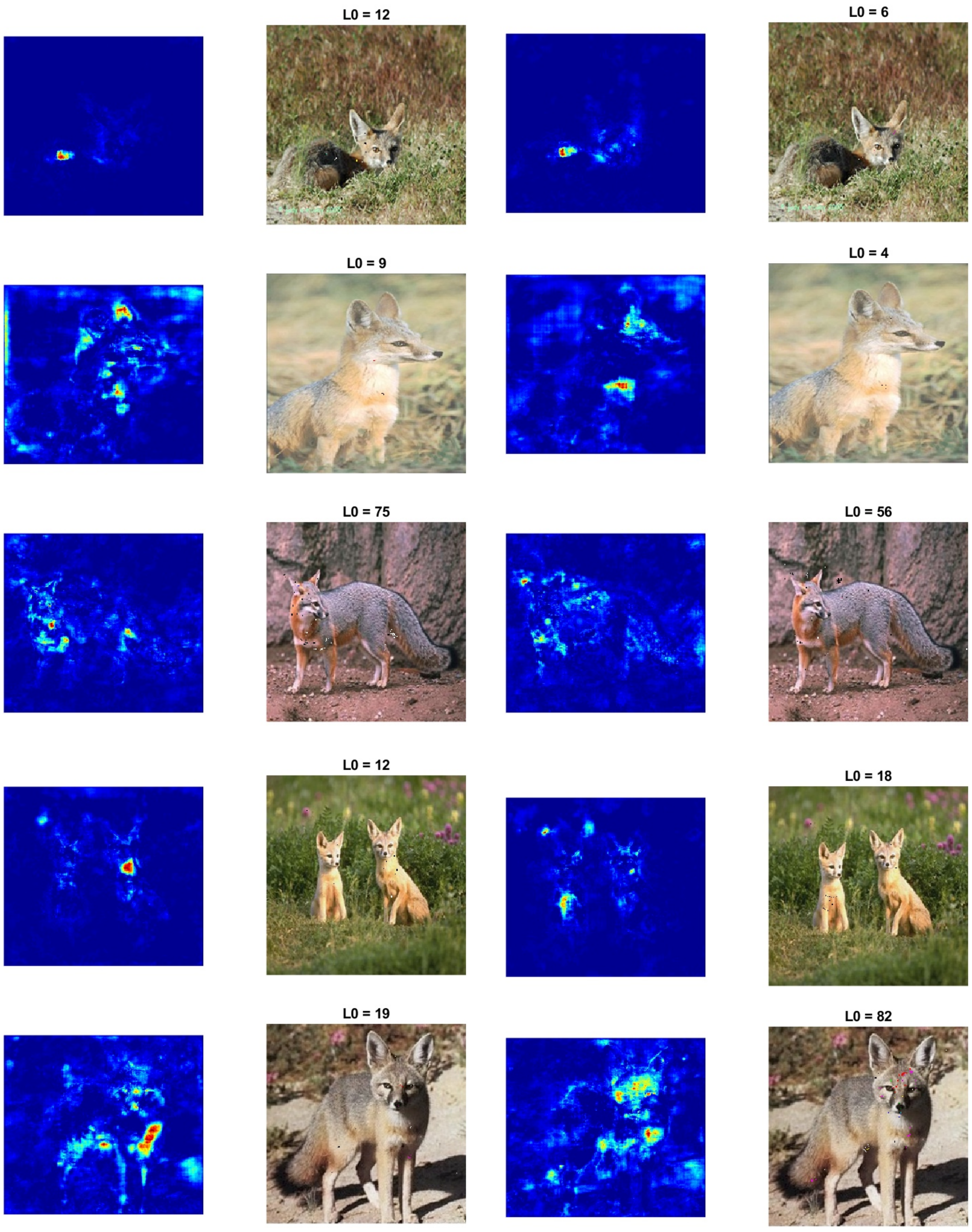


Figure A.3: Adversarial examples on upper boundaries (right column) returned by DeepTRE, and their saliency maps (left column) for the ImageNet VGG-16 and VGG-19 models. The first and second columns are for the VGG-16 model; the third and fourth columns are for the VGG-19 model.

A.2 Local/Global Robustness and Convergence

A.2.1 Hardware and Software Platforms

- Hardware Platform:
  - NVIDIA GeForce GTX 1050
  - Intel® Core™ i7-7700HQ Processor
- Software Platform:
  - MATLAB 2018a
  - Neural Network Toolbox
  - Image Processing Toolbox
  - Parallel Computing Toolbox

A.2.2 DNN-Reduced: Model Structure and Parameter Setting

The model structure of DNN-Reduced is in Table A.1.

Table A.1: DNN-Reduced

Layer Type	Size
Input	$14 \times 14 \times 1$
Convolution + Batch Normalisation + ReLU	$2 \times 2 \times 8$
Convolution + Batch Normalisation + ReLU	$2 \times 2 \times 16$
Convolution + Batch Normalisation + ReLU	$2 \times 2 \times 32$
Fully Connected + Softmax	10

DNN-Reduced: Training Setup

- Parameter Optimisation Option:
  - Mini-Batch Size = 128
  - Max Epochs = 50

  - Optimizer = `SGDM`

- Training Accuracy:
  - MNIST (`99.5%` on 50 000 images)
- Testing Accuracy:
  - MNIST (`98.73%` on 10 000 images)

DeepTRE: Parameter Setting

- EPSILON = `0.25`
- Subspace_Dimension = `3`
- Tested Images: 5300 `MNIST Test Images`

A.2.3 DNN-Standard: Model Structure and Parameter Setting

The model structure of DNN-Standard is in Table A.2.

Table A.2: DNN-Standard

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 32$
Convolution + ReLU	$3 \times 3 \times 64$
Max-Pooling	2×2
Dropout	0.25
Fully Connected + ReLU	128
Dropout	0.5
Fully Connected + Softmax	10

DNN-Standard: Training Setup

- Parameter Optimisation Option:

  - Mini-Batch Size = `128`
  - Max Epochs = `30`
  - Optimizer = `SGDM`

- Training Accuracy:

  - MNIST (`100%` on 50 000 images)

- Testing Accuracy:

  - MNIST (`99.16%` on 10 000 images)

DeepTRE: Parameter Setting

- EPSILON = `0.25`
- Subspace_Dimension = `2`
- Tested Images: 2400 `MNIST Test Images`

A.2.4 Ground-Truth Adversarial Images

Figures A.4 and A.5 display some adversarial images returned by our tool DeepTRE.

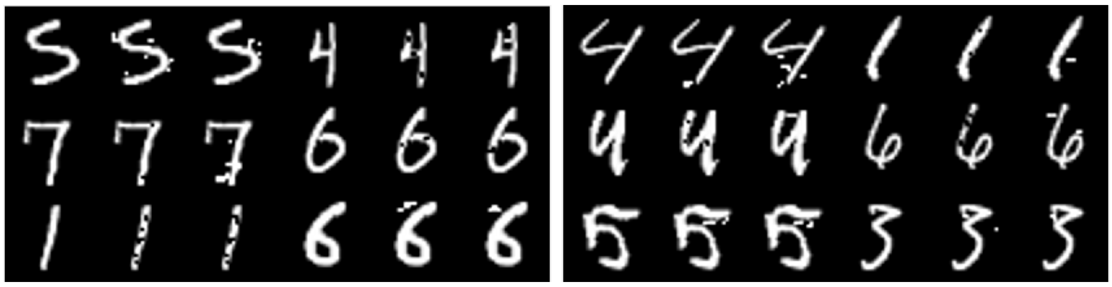


Figure A.4: Ground truth adversarial examples when converging to `MSR`. For each digital image, from the left to right, the first is original image, the second is the adversarial image returned at $t = 1$, and the third is the adversarial example at the boundary of a safe norm ball, namely the ground-truth adversarial example [7].

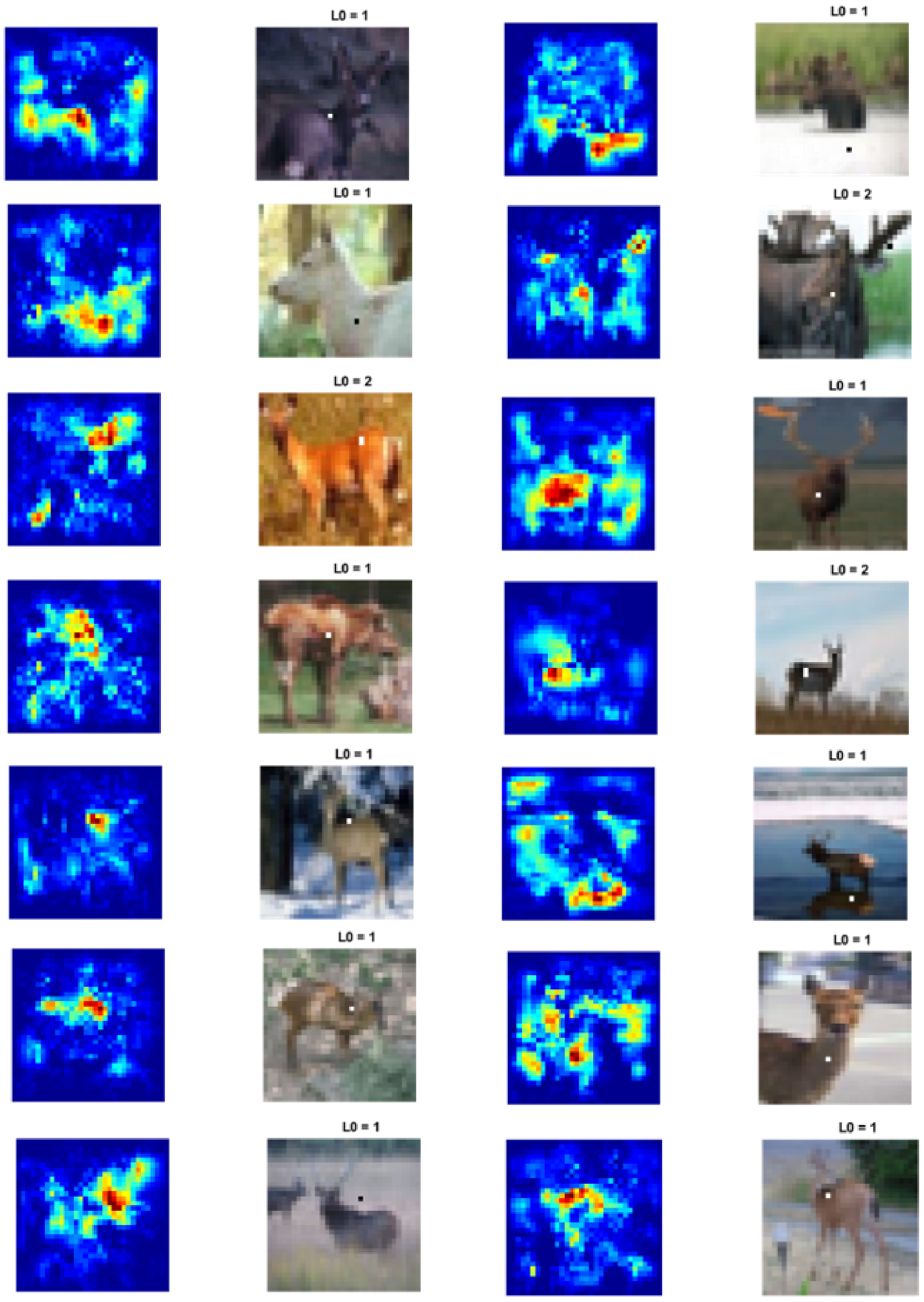


Figure A.5: Ground-truth adversarial examples (right column) generated by DeepTRE at $t = 1$, and the saliency maps of the original images (left column).

A.3 Robustness and Accuracy of Model Architecture

A.3.1 Hardware and Software Platforms

- Hardware Platform:
  - NVIDIA GeForce GTX 1050
  - Intel® Core™ i7-7700HQ Processor
- Software Platform:
  - MATLAB 2018a
  - Neural Network Toolbox
  - Image Processing Toolbox
  - Parallel Computing Toolbox

A.3.2 DNN-1 to DNN-7: Model Structure and Parameter Setting

The model structures of DNN-1 to DNN-7 are presented from Tables A.3 to A.9.

DNN-1 to DNN-7: Training Setup

- Parameter Optimisation Option:
  - Mini-Batch Size = `128`
  - Max Epochs = `30`
  - Optimizer = `SGDM`
- Training Accuracy of MNIST (50 000 images):
  - DNN-1 to DNN-7 all reach `100%`
- Testing Accuracy of MNIST (10 000 images):
  - DNN-1 = `97.75%`
  - DNN-2 = `97.95%`
  - DNN-3 = `98.38%`
  - DNN-4 = `99.06%`

- DNN-5 = `99.16%`
- DNN-6 = `99.13%`
- DNN-7 = `99.41%`

DeepTRE: Parameter Setting

- EPSILON = `0.3`
- Subspace_Dimension = `2`
- Tested Images: 1000 `MNIST Test Images`

Table A.3: DNN-1

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 32$
Fully Connected + Softmax	10

Table A.4: DNN-2

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 32$
Convolution + ReLU	$3 \times 3 \times 64$
Fully Connected + Softmax	10

Table A.5: DNN-3

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 32$
Convolution + Batch Normalisation + ReLU	$3 \times 3 \times 64$
Fully Connected + Softmax	10

Table A.6: DNN-4

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 32$
Convolution + Batch Normalisation + ReLU	$3 \times 3 \times 64$
Fully Connected + ReLU	128
Fully Connected + Softmax	10

Table A.7: DNN-5

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 32$
Convolution + Batch Normalisation + ReLU	$3 \times 3 \times 64$
Dropout	0.5
Fully Connected + ReLU	128
Fully Connected + Softmax	10

Table A.8: DNN-6

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 32$
Convolution + ReLU	$3 \times 3 \times 64$
Convolution + Batch Normalisation + ReLU	$3 \times 3 \times 128$
Dropout	0.5
Fully Connected + ReLU	128
Fully Connected + Softmax	10

Table A.9: DNN-7

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 16$
Convolution + Batch Normalisation + ReLU	$3 \times 3 \times 32$
Convolution + Batch Normalisation + ReLU	$3 \times 3 \times 64$
Convolution + Batch Normalisation + ReLU	$3 \times 3 \times 128$
Dropout	0.5
Fully Connected + ReLU	256
Dropout	0.5
Fully Connected + Softmax	10

B

Proofs and Comparisons for DeepGame

B.1 Proofs for Lemmas and Theorems

B.1.1 Proof for Lemma 4

Proof. Let $\boldsymbol{\alpha}_1$ be any point in $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$. We need to show that, for some τ-grid input $\boldsymbol{\alpha}'$, we have $\boldsymbol{\alpha}_1 \in \mathsf{Ball}(\boldsymbol{\alpha}', L^p, \frac{1}{2}d(L^p, \tau))$. Because every point in $\mathsf{Ball}(\boldsymbol{\alpha}, L^p, d)$ belongs to a τ-grid cell, we assume that $\boldsymbol{\alpha}_1$ is in a τ-grid cell which, without loss of generality, has a set T of τ-grid inputs as its vertices. Now for any two τ-grid inputs $\boldsymbol{\alpha}_2$ and $\boldsymbol{\alpha}_3$ in T, we have $\|\boldsymbol{\alpha}_2 - \boldsymbol{\alpha}_3\|_p \leq d(L^p, \tau)$ by the construction of the grid. Therefore, we have $\boldsymbol{\alpha}_1 \in \mathsf{Ball}(\boldsymbol{\alpha}', L^p, \frac{1}{2}d(L^p, \tau))$ for some $\boldsymbol{\alpha}' \in T$. $\square$

B.1.2 Proof for Lemma 5

Proof. We prove by contradiction. Assume that $\texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) = d'$ for some $d' > 0$, and $\texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) < d' - \frac{1}{2}d(L^p, \tau)$. Then there must exist an input $\boldsymbol{\alpha}'$ such that $\boldsymbol{\alpha}' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$ and

$$\|\boldsymbol{\alpha}' - \boldsymbol{\alpha}\|_p = \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) < d' - \frac{1}{2}d(L^p, \tau), \tag{B.1}$$

and $\boldsymbol{\alpha}'$ is not a τ-grid input. By Lemma 4, there must exist a τ-grid input

$\boldsymbol{\alpha}''$ such that $\boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha}'', L^p, \frac{1}{2}d(L^p, \tau))$. Now because all τ-grid inputs are misclassification aggregators with respect to $\frac{1}{2}d(L^p, \tau)$, we have $\boldsymbol{\alpha}'' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$. By $\boldsymbol{\alpha}'' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$ and the fact that $\boldsymbol{\alpha}''$ is a τ-grid input, we have that

$$\texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) \leq \|\boldsymbol{\alpha} - \boldsymbol{\alpha}''\|_p \leq \|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_p + \frac{1}{2}d(L^p, \tau). \tag{B.2}$$

Now, combining Equations (B.1) and (B.2), we have $\texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) < d'$, which contradicts the hypothesis that $\texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) = d'$. □

B.1.3 Proof for Theorem 6

Proof. By Lemma 3, we have $\texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) \leq \texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau)$ for any $\tau > 0$. By Lemmas 5 and 6, when $\texttt{FMSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) = d'$, we have $\texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) \geq d' - \frac{1}{2}d(L^p, \tau)$, under the condition that

$$d(L^p, \tau) \leq \frac{2 \cdot \mathsf{Margin}(\boldsymbol{\alpha}', \mathcal{N}(\boldsymbol{\alpha}'))}{\max_{c \in C, c \neq \mathcal{N}(\boldsymbol{\alpha}')}(\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')} + \mathsf{Lip}_c)} \tag{B.3}$$

for all τ-grid inputs $\boldsymbol{\alpha}' \in \Gamma(\boldsymbol{\alpha}, L^p, d, \tau)$. Therefore, when $d(L^p, \tau)$ satisfies the above condition for all τ-grid inputs $\boldsymbol{\alpha}' \in \Gamma(\boldsymbol{\alpha}, L^p, d, \tau)$, if we use d' to estimate $\texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d)$, we will have $d' - \frac{1}{2}d(L^p, \tau) \leq \texttt{MSR}(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) \leq d'$, i.e., the error bound is $\frac{1}{2}d(L^p, \tau)$. □

B.1.4 Proof for Lemma 7

Proof. We prove by contradiction. Assume that $\texttt{FFR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d, \tau) = d'$ for some $d' > 0$, and $\texttt{FR}_\Lambda(\mathcal{N}, \boldsymbol{\alpha}, c, L^p, d) < d' - \frac{1}{2}d(L^p, \tau)$. Then, for all subsets $\Lambda \subseteq \Lambda(\boldsymbol{\alpha})$ of features, either (1) for all $X \subseteq \bigcup_{\lambda \in \Lambda} P_\lambda$ and $\psi \in \Psi$ we have $\delta_{\tau,X,\psi}(\boldsymbol{\alpha}) \notin \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$, or (2) there must exist $X \subseteq \bigcup_{\lambda \in \Lambda} P_\lambda$ and $\psi \in \Psi$ such that $\boldsymbol{\alpha}' = \delta_{\tau,X,\psi}(\boldsymbol{\alpha}) \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha}, c)$ and

$$\|\boldsymbol{\alpha}' - \boldsymbol{\alpha}\|_p < d' - \frac{1}{2}d(L^p, \tau), \tag{B.4}$$

and $\boldsymbol{\alpha}'$ is not a τ-grid input. For the latter case, by Lemma 4, there must exist a τ-grid input $\boldsymbol{\alpha}''$ such that $\boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha}'', L^p, \frac{1}{2}d(L^p, \tau))$. Now because all τ-grid inputs are misclassification aggregators with respect to $\frac{1}{2}d(L^p,\tau)$, we have $\boldsymbol{\alpha}'' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha},c)$. By $\boldsymbol{\alpha}'' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha},c)$ and the fact that $\boldsymbol{\alpha}''$ is a τ-grid input, we have that

$$\left\|\boldsymbol{\alpha}-\boldsymbol{\alpha}''\right\|_p \leq \left\|\boldsymbol{\alpha}-\boldsymbol{\alpha}'\right\|_p + \frac{1}{2}d(L^p,\tau). \tag{B.5}$$

Hence, we have $\mathtt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau) < d'$ by combining Equations (B.4) and (B.5). However, this contradicts the hypothesis that $\mathtt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau) = d'$. As for the former case, we have that $\mathtt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau) = d' > d$. If $\mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d) < d' - \frac{1}{2}d(L^p,\tau)$, then there exists an $\boldsymbol{\alpha}'$ such that $\boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha}'',L^p,\frac{1}{2}d(L^p,\tau))$ for some τ-grid input $\boldsymbol{\alpha}''$. By the definition of misclassification aggregator, we have $\boldsymbol{\alpha}'' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha},c)$. This contradicts the hypothesis that $\mathtt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau) = d' > d$. □

B.1.5 Proof for Theorem 7

Proof. By Lemma 3, we have $\mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d) \leq \mathtt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau)$ for any $\tau > 0$. By Lemma 6 and Lemma 7, when $\mathtt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau) = d'$, we have $\mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d) \geq d' - \frac{1}{2}d(L^p,\tau)$, under the condition that

$$d(L^p,\tau) \leq \frac{2\cdot\mathsf{Margin}(\boldsymbol{\alpha}',\mathcal{N}(\boldsymbol{\alpha}'))}{\max_{c\in C,c\neq\mathcal{N}(\boldsymbol{\alpha}')}(\mathsf{Lip}_{\mathcal{N}(\boldsymbol{\alpha}')}+\mathsf{Lip}_c)} \tag{B.6}$$

for all τ-grid inputs $\boldsymbol{\alpha}' \in \Gamma(\boldsymbol{\alpha},L^p,d,\tau)$. Therefore, when $d(L^p,\tau)$ satisfies the above condition for all τ-grid inputs $\boldsymbol{\alpha}' \in \Gamma(\boldsymbol{\alpha},L^p,d,\tau)$, if we use d' to estimate $\mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)$, we will have $d' - \frac{1}{2}d(L^p,\tau) \leq \mathtt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d) \leq d'$, i.e., the error bound is $\frac{1}{2}d(L^p,\tau)$. □

B.1.6 Proof for Theorem 8

Proof. First, we show that $\left\|\boldsymbol{\alpha}-\boldsymbol{\alpha}'\right\|_p \geq val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\mathtt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d))$ for any input $\boldsymbol{\alpha}'$ such that $\boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha},L^p,d)$, $\boldsymbol{\alpha}' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha},c)$, and $\boldsymbol{\alpha}'$ is a τ-grid input. Intuitively, it says that Player $\mathtt{I}$ reward from the game on the initial state s_0 is no greater than the distance to any τ-grid adversarial example.

That is, once computed, the $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\texttt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d))$ is a lower bound of the optimisation $\texttt{FMSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau)$. This can be obtained by the fact that every τ-grid input can be reached by some game play.

Second, from the termination condition of the game plays, we can see that if $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\texttt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)) \leq \|\boldsymbol{\alpha}-\boldsymbol{\alpha}'\|_p$ for some $\boldsymbol{\alpha}'$ then there must exist some $\boldsymbol{\alpha}''$ such that $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\texttt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)) = \|\boldsymbol{\alpha}-\boldsymbol{\alpha}''\|_p$. Therefore, we have that $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\texttt{MSR}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d))$ is the minimum value of $\|\boldsymbol{\alpha}-\boldsymbol{\alpha}'\|_p$ among all $\boldsymbol{\alpha}'$ with $\boldsymbol{\alpha}' \in \mathsf{Ball}(\boldsymbol{\alpha},L^p,d)$, $\boldsymbol{\alpha}' \in \mathsf{adv}_{L^p,d}(\boldsymbol{\alpha},c)$, and $\boldsymbol{\alpha}'$ is a τ-grid input.

Finally, we notice that the above minimum value of $\|\boldsymbol{\alpha}-\boldsymbol{\alpha}'\|_p$ is equivalent to the optimal value required by Equation (5.6). □

B.1.7 Proof for Theorem 9

Proof. First of all, let Λ_1 be the set of features and Δ_1 be the set of atomic input manipulations in achieving the optimal value of $\texttt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau)$. We can construct a game play for (Λ_1,Δ_1). More specifically, the game play leads from the initial state to a terminal state, by recursively selecting an unused input manipulation and its associated feature and defining the corresponding moves for Player I and Player II, respectively. Therefore, because the strategy profile σ is optimal, we have $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\texttt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)) \geq \texttt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau)$.

On the other hand, we notice that the ordering of the applications of atomic input manipulations does not matter, because the reward of the terminal state is the distance from its associated input to the original input. Therefore, because the game explores exactly all the possible applications of atomic input manipulations and $\texttt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau)$ is the optimal value by its definition, by Lemma 2 we have that $val(\mathcal{G}(\mathcal{N},\boldsymbol{\alpha},c,L^p,d),\texttt{FR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d)) \leq \texttt{FFR}_\Lambda(\mathcal{N},\boldsymbol{\alpha},c,L^p,d,\tau)$. □

B.2 Comparison of Tools in Adversarial Attacks

B.2.1 Model Architectures and Accuracy Rates

The architectures of the MNIST and CIFAR-10 models used in adversarial attacks are illustrated in Table B.1. Below we list the parameters during the training process together with the training and testing accuracy rates.

Table B.1: Architectures of the MNIST, CIFAR-10, and GTSRB models

Layer Type	MNIST	CIFAR-10	GTSRB
Input	$28 \times 28 \times 1$	$32 \times 32 \times 3$	$48 \times 48 \times 3$
Convolution + ReLU	$3 \times 3 \times 32$	$3 \times 3 \times 64$	$3 \times 3 \times 64$
Convolution + ReLU	$3 \times 3 \times 32$	$3 \times 3 \times 64$	$3 \times 3 \times 64$
Max-Pooling	2×2	2×2	2×2
Convolution + ReLU	$3 \times 3 \times 64$	$3 \times 3 \times 128$	$3 \times 3 \times 128$
Convolution + ReLU	$3 \times 3 \times 64$	$3 \times 3 \times 128$	$3 \times 3 \times 128$
Max-Pooling	2×2	2×2	2×2
Flatten			
Fully Connected + ReLU	200	256	256
Dropout	0.5	0.5	0.5
Fully Connected + ReLU	200	256	256
Fully Connected + Softmax	10	10	43

- Parameter Optimisation Option:
  - Batch Size = `128`
  - Epochs = `50`
  - Loss Function = `softmax_cross_entropy_with_logits`
  - Optimizer = `SGD` (lr=`0.01`, decay=`1e-6`, momentum=`0.9`, nesterov=`True`)
- Training Accuracy:

  - MNIST (`99.99%` on 60 000 images)
  - CIFAR-10 (`99.83%` on 50 000 images)

- Testing Accuracy:
  - MNIST (`99.36%` on 10 000 images)
  - CIFAR-10 (`78.30%` on 10 000 images)

B.2.2 Parameter Settings of Existing Tools

We remark that, unless separately specified, we employ the same parameter setting of each tool to generate adversarial examples on the MNIST and CIFAR-10 datasets.

- DeepTRE:
  - EPSILON = `0.5`
  - L0_UPPER_BOUND = `100`
- DeepGame
  - gameType = `cooperative`
  - bound = `ub`
  - algorithm = `A*`
  - eta = `(L0, 30)`
  - tau = `1`
- C&W:
  - targeted = `False`
  - learning_rate = `0.1`
  - max_iteration = `100`
- DLV:
  - mcts_mode = `sift_twoPlayer`
  - startLayer, maxLayer = `-1`
  - numOfFeatures = `150`
  - featureDims = `1`

  - MCTS_level_maximal_time = `30`
  - MCTS_all_maximal_time = `120`
  - MCTS_multi_samples = `5` (MNIST), `3` (CIFAR-10)

- SafeCV:
  - MANIP = `max_manip` (MNIST), `white_manipulation` (CIFAR-10)
  - VISIT_CONSTANT = `1`
  - backtracking_constant = `1`
  - simulation_cutoff = `75` (MNIST), `100` (CIFAR-10)
  - small_image = `True`

- JSMA:
  - bounds = `(0, 1)`
  - predicts = `logits`

B.2.3 Hardware and Software Platforms

- Hardware Platform:
  - NVIDIA GeForce GTX TITAN Black
  - Intel® Core™ i5-4690S CPU @ 3.20GHz $\times$ 4

- Software Platform:
  - Ubuntu 14.04.3 LTS
  - Fedora 26 (64-bit)
  - Anaconda, PyCharm

B.2.4 Adversarial Images

Figures B.1 and B.2 present a few adversarial examples generated on the MNIST and CIFAR-10 datasets by our tools DeepTRE and DeepGame in comparison to some other existing approaches. Figure B.3 exhibits some adversarial examples generated by our tool DeepGame on the GTSRB dataset.

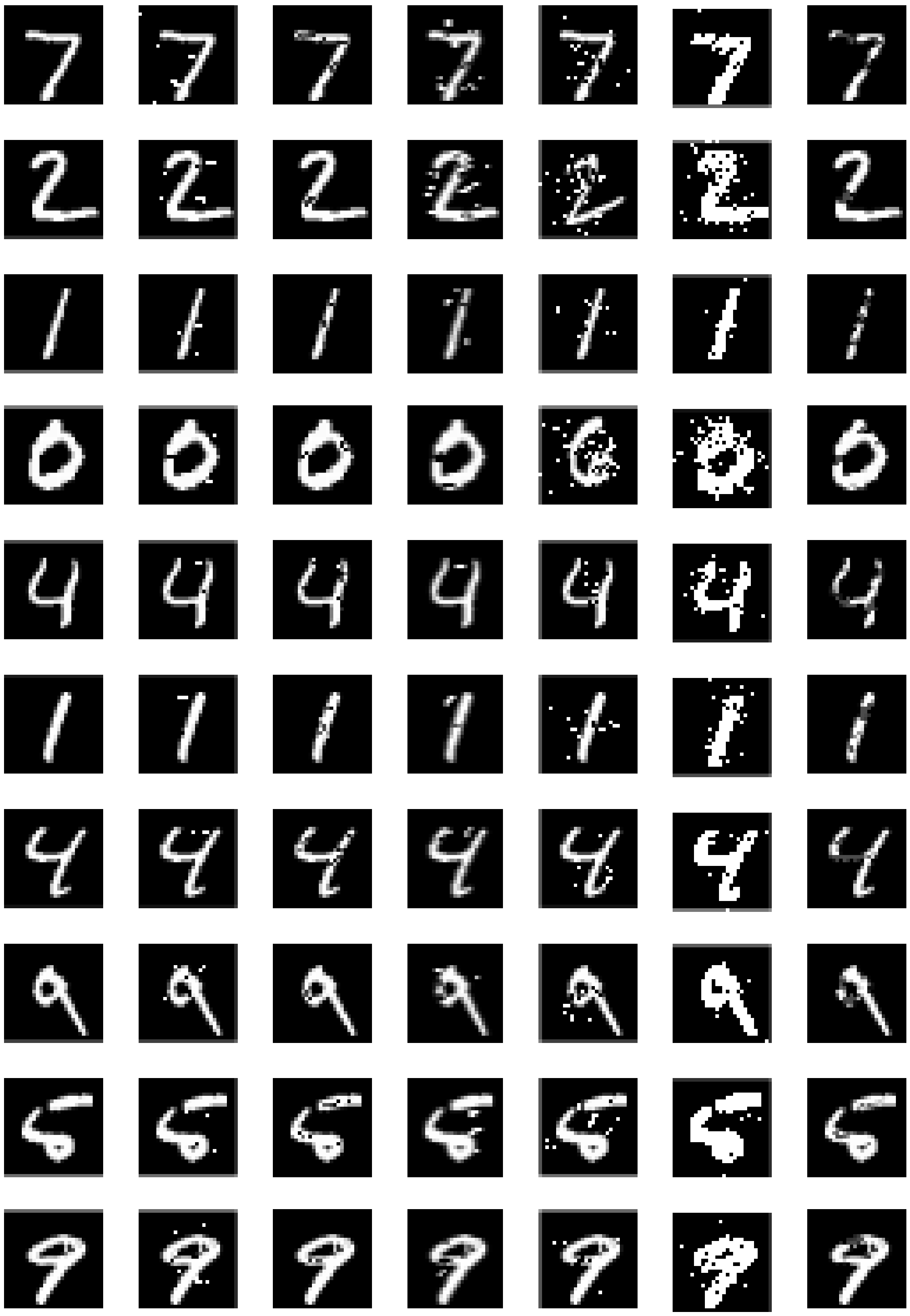


Figure B.1: Comparison of the generated adversarial MNIST images. From left to right: the original image, DeepGame, C&W, DeepTRE, DLV, SafeCV, and JSMA.

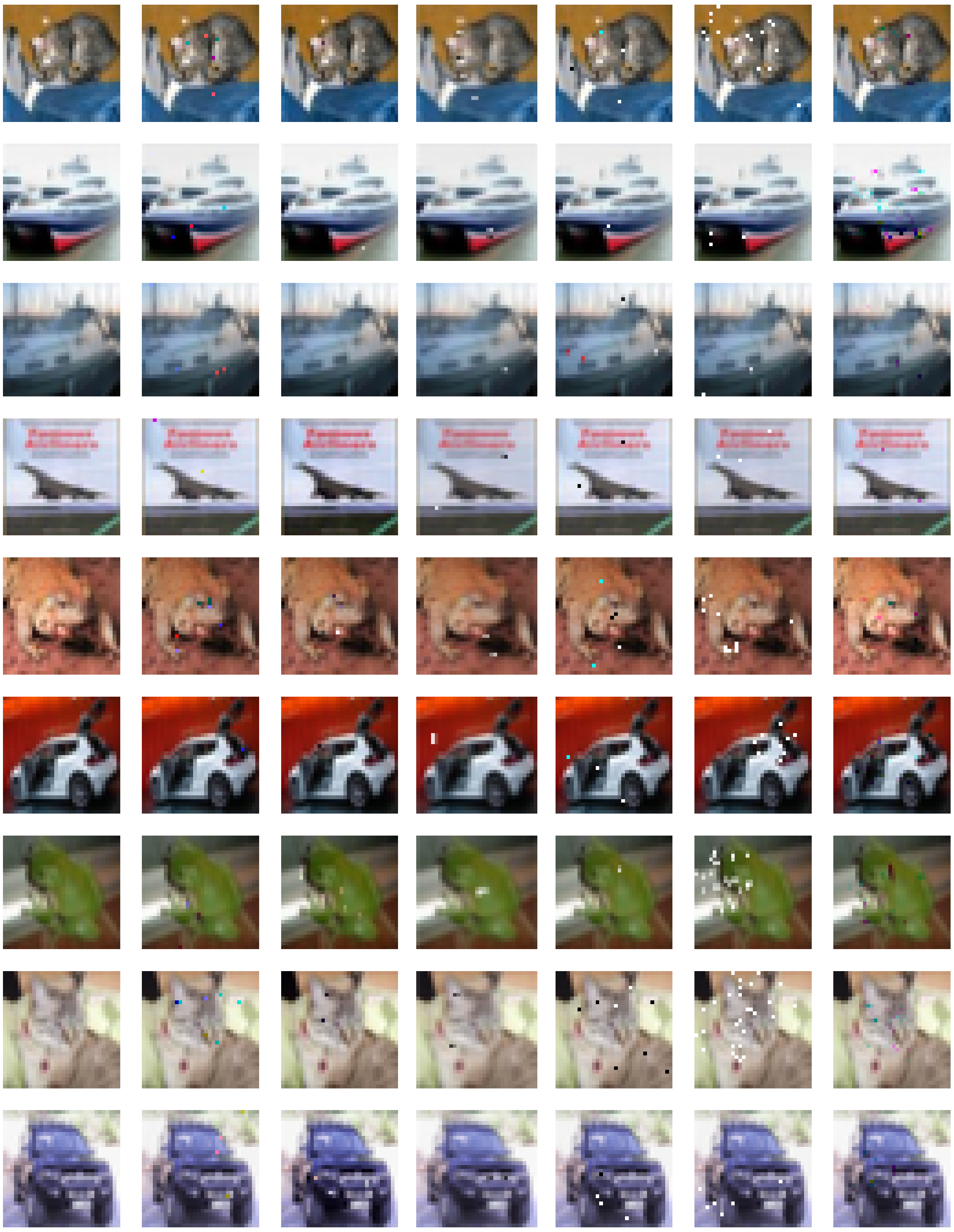


Figure B.2: Comparison of generated adversarial CIFAR-10 images. From left to right: the original image, DeepGame, C&W, DeepTRE, DLV, SafeCV, and JSMA.

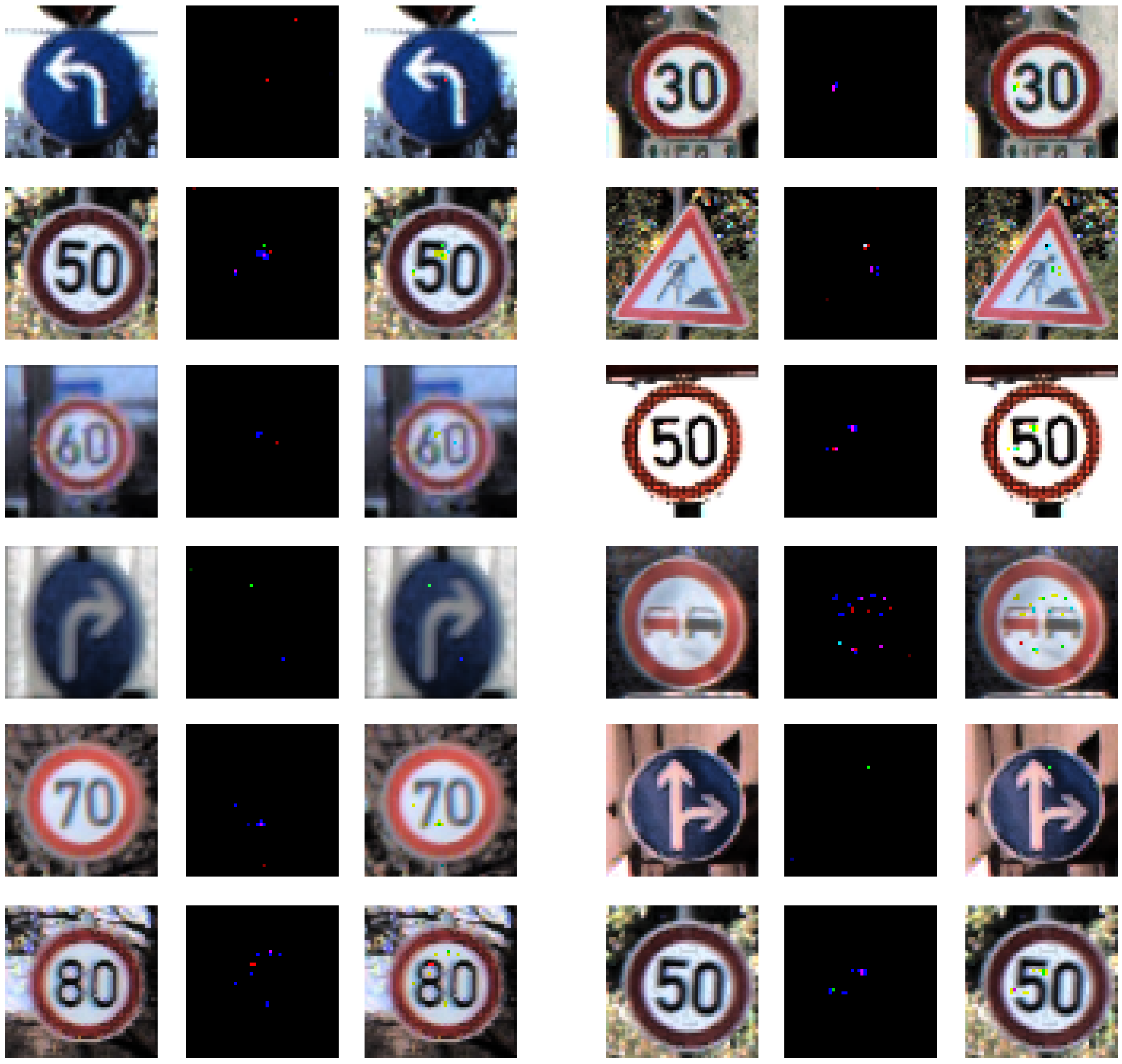


Figure B.3: Adversarial GTSRB images generated by tool DeepGame.

Bibliography

[1] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. “Measuring Neural Net Robustness with Constraints”. In: *Advances in Neural Information Processing Systems 29.* Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 2016, pp. 2613–2621.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. “Speeded-Up Robust Features (SURF)”. In: *Computer Vision and Image Understanding* 110.3 (2008). Similarity Matching in Computer Vision and Multimedia, pp. 346–359.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “SURF: Speeded Up Robust Features”. In: *Computer Vision – ECCV 2006.* Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[4] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. “Evasion Attacks against Machine Learning at Test Time”. In: *Machine Learning and Knowledge Discovery in Databases.* Ed. by Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 387–402.

[5] Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. “A unified view of piecewise linear neural network verification”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems.* 2018, pp. 4795–4804.

[6] Andrew Burton and John Radford. *Thinking in Perspective: Critical Essays in the Study of Thought Processes.* Vol. 646. Routledge, 1978.

[7] Nicholas Carlini, Guy Katz, Clark W. Barrett, and David L. Dill. “Ground-Truth Adversarial Examples”. In: *CoRR* abs/1709.10207 (2017). arXiv: `1709.10207`.

[8] Nicholas Carlini and David Wagner. “Towards Evaluating the Robustness of Neural Networks”. In: *2017 IEEE Symposium on Security and Privacy (SP).* May 2017, pp. 39–57.

[9] Gunnar Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian. “On the Local Behavior of Spaces of Natural Images”. In: *International Journal of Computer Vision* 76.1 (Jan. 2008), pp. 1–12.

[10] G.M.J.B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, and B. Bouzy. “Progressive Strategies for Monte-Carlo Tree Search”. In: *New Mathematics and Natural Computation* 4.3 (2008), pp. 343–359.

[11] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. “Maximum Resilience of Artificial Neural Networks”. In: *Automated Technology for Verification and Analysis*. Ed. by Deepak D’Souza and K. Narayan Kumar. Cham: Springer International Publishing, 2017, pp. 251–268.

[12] Patrick Cousot and Radhia Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. In: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. Association for Computing Machinery, 1977, pp. 238–252.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, pp. 248–255.

[14] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. “Output Range Analysis for Deep Feedforward Neural Networks”. In: *NASA Formal Methods*. Ed. by Aaron Dutle, César Muñoz, and Anthony Narkawicz. Cham: Springer International Publishing, 2018, pp. 121–138.

[15] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. “A Dual Approach to Scalable Verification of Deep Networks.” In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2018, pp. 550–559.

[16] Rüdiger Ehlers. “Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks”. In: *Automated Technology for Verification and Analysis*. Ed. by Deepak D’Souza and K. Narayan Kumar. Cham: Springer International Publishing, 2017, pp. 269–286.

[17] Gunnar Farnebäck. “Two-Frame Motion Estimation Based on Polynomial Expansion”. In: *Image Analysis*. Ed. by Josef Bigun and Tomas Gustavsson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370.

[18] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. “AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. May 2018, pp. 3–18.

[19] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323.

[20] Ian J. Goodfellow. “Gradient Masking Causes CLEVER to Overestimate Adversarial Perturbation Size”. In: *CoRR* abs/1804.07870 (2018). arXiv: `1804.07870`.

[21] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* The MIT Press, 2016.

[23] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27.* Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2016, pp. 770–778.

[25] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[26] Berthold K.P. Horn and Brian G. Schunck. "Determining Optical Flow". In: *Artificial Intelligence* 17.1 (1981), pp. 185–203.

[27] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. "A Survey of Safety and Trustworthiness of Deep Neural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability". In: *Computer Science Review* 37 (2020), p. 100270.

[28] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. "Safety Verification of Deep Neural Networks". In: *Computer Aided Verification.* Ed. by Rupak Majumdar and Viktor Kunčak. Cham: Springer International Publishing, 2017, pp. 3–29.

[29] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. "What is the Best Multi-Stage Architecture for Object Recognition?" In: *2009 IEEE 12th International Conference on Computer Vision.* Sept. 2009, pp. 2146–2153.

[30] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *Computer Aided Verification.* Ed. by Rupak Majumdar and Viktor Kunčak. Cham: Springer International Publishing, 2017, pp. 97–117.

[31] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.* Ed. by Yoshua Bengio and Yann LeCun. 2015.

[32] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *Machine Learning: ECML 2006.* Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293.

[33] Alex Krizhevsky and Geoffrey Hinton. "Learning Multiple Layers of Features from Tiny Images". In: Technical Report, University of Toronto, 2009.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.

[35] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. “Adversarial Examples in the Physical World”. In: *Artificial Intelligence Safety and Security*. Chapman and Hall/CRC, 2018, pp. 99–112.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[37] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324.

[38] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. “Analyzing Deep Neural Networks with Symbolic Propagation: Towards Higher Precision and Faster Verification”. In: *Static Analysis*. Ed. by Bor-Yuh Evan Chang. Cham: Springer International Publishing, 2019, pp. 296–319.

[39] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. “Algorithms for Verifying Deep Neural Networks”. In: *Foundations and Trends® in Optimization* 4.3-4 (2021), pp. 244–404. URL: `http://dx.doi.org/10.1561/2400000035`.

[40] Alessio Lomuscio and Lalit Maganti. “An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks”. In: *CoRR* abs/1706.07351 (2017). arXiv: `1706.07351`.

[41] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110.

[42] Bruce D. Lucas and Takeo Kanade. “An Iterative Image Registration Technique with an Application to Stereo Vision”. In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI’81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[43] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 4765–4774.

[44] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[45] Matthew Mirman, Timon Gehr, and Martin Vechev. “Differentiable Abstract Interpretation for Provably Robust Neural Networks”. In: *Proceedings of the 35th International Conference on Machine Learning.* Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 3578–3586.

[46] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. “Universal Adversarial Perturbations”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* July 2017, pp. 86–94.

[47] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. “DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2016, pp. 2574–2582.

[48] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10).* Ed. by Johannes Fürnkranz and Thorsten Joachims. Haifa, Israel: Omnipress, June 2010, pp. 807–814.

[49] Nina Narodytska. “Formal Analysis of Deep Binarized Neural Networks”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18.* International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 5692–5696.

[50] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. “Verifying Properties of Binarized Deep Neural Networks”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18).* 2018, pp. 6615–6624.

[51] Arnold Neumaier and Oleg Shcherbina. “Safe Bounds in Linear and Mixed-Integer Linear Programming”. In: *Mathematical Programming* 99.2 (2004), pp. 283–296.

[52] Mıcheál O’Searcoid. *Metric Spaces.* Springer, 2007.

[53] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. “The Limitations of Deep Learning in Adversarial Settings”. In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P).* Mar. 2016, pp. 372–387.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32.* Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8026–8037.

[55] Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. “Lower bounds on the robustness to adversarial perturbations”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 804–813.

[56] Luca Pulina and Armando Tacchella. “An Abstraction-Refinement Approach to Verification of Artificial Neural Networks”. In: *Computer Aided Verification*. Ed. by Tayssir Touili, Byron Cook, and Paul Jackson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 243–257.

[57] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. “Certified Defenses against Adversarial Examples”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[58] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “"Why Should I Trust You?": Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 1135–1144.

[59] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. “Reachability Analysis of Deep Neural Networks with Provable Guarantees”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 2651–2659.

[60] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. “Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for the Hamming Distance”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 5944–5952.

[61] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning Representations by Back-Propagating Errors”. In: *Nature* 323.6088 (1986), pp. 533–536.

[62] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[63] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009.

[64] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[65] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild". In: *CRCV-TR-12-01.* November, 2012.

[66] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition". In: *Neural Networks* 32 (2012). Selected Papers from IJCNN 2011, pp. 323–332.

[67] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. "Concolic Testing for Deep Neural Networks". In: *Automated Software Engineering (ASE).* ACM, 2018, pp. 109–119.

[68] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. "Intriguing Properties of Neural Networks". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.* 2014.

[69] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. "Robustness May Be at Odds with Accuracy". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net, 2019.

[70] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. "Formal Security Analysis of Neural Networks using Symbolic Intervals". In: *27th USENIX Security Symposium (USENIX Security 18).* Baltimore, MD: USENIX Association, Aug. 2018, pp. 1599–1614.

[71] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. "Multiscale Structural Similarity for Image Quality Assessment". In: *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003.* Vol. 2. Nov. 2003, pp. 1398–1402.

[72] David H Warren and Edward R Strelow. *Electronic Spatial Sensing for the Blind: Contributions from Perception, Rehabilitation, and Computer Vision.* Vol. 99. Springer Science & Business Media, 1985.

[73] Shang-Chia Wei and Tso-Jung Yen. "Superpixels Generating from the Pixel-based K-Means Clustering." In: *Journal of Multimedia Processing and Technologies* 6.3 (2015), pp. 77–86.

[74] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. "Towards Fast Computation of Certified Robustness for ReLU Networks". In: *Proceedings of the 35th International Conference on Machine Learning.* Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 5276–5285.

[75] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. "Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.* OpenReview.net, 2018.

[76] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. "Feature-Guided Black-Box Safety Testing of Deep Neural Networks". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Dirk Beyer and Marieke Huisman. Cham: Springer International Publishing, 2018, pp. 408–426.

[77] Eric Wong and Zico Kolter. "Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 5286–5295.

[78] Min Wu and Marta Kwiatkowska. "Robustness Guarantees for Deep Neural Networks on Videos". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 308–317.

[79] Min Wu, Tyron Louw, Morteza Lahijanian, Wenjie Ruan, Xiaowei Huang, Natasha Merat, and Marta Kwiatkowska. "Gaze-based Intention Anticipation over Driving Manoeuvres in Semi-Autonomous Vehicles". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2019, pp. 6210–6216.

[80] Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. "A Game-Based Approximate Verification of Deep Neural Networks with Provable Guarantees". In: *Theoretical Computer Science* 807 (2020). In memory of Maurice Nivat, a founding father of Theoretical Computer Science - Part II, pp. 298–329.

[81] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. "Output Reachable Set Estimation and Verification for Multilayer Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.11 (Nov. 2018), pp. 5777–5783.

[82] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. "Efficient Neural Network Robustness Certification with General Activation Functions". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 4939–4948.

[83] Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. "RecurJac: An Efficient Recursive Algorithm for Bounding Jacobian Matrix of Neural Networks and Its Applications". In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. Vol. 33. 2019, pp. 5757–5764.