Safety Evaluation of Attention Models for Image Classification Problems

Xin Zhang

Supervisor: Marta Kwiatkowska

Honour School of Computer Science - Part B

Trinity, 2022

Word count: 9,385

Abstract

Recent development in machine learning has led to models with state-of-the-art performance in areas like computer vision. In particular, there is growing interest in using deep neural networks for safety-critical applications. The robustness of a neural network model, which measures the model's capability to produce the correct output despite small perturbations in the input, is crucial for such applications.

There are various definitions of robustness. In this project, we use an existing tool called `DeepGame` that bounds the pointwise robustness of neural networks. Given a correctly classified input, the pointwise robustness measures how much we need to perturb the input for the model to misclassify it.

`DeepGame` has been used on pure convolutional neural networks (CNNs). Other neural network architectures, such as CNNs with attention mechanisms, have worked well in image classification tasks. In this project, we select two kinds of attention mechanisms from existing literature and implement them, train CNNs with and without the attention mechanisms for image classification, and use `DeepGame` to bound the robustness of the resulting models. Moreover, we improve `DeepGame` for greater efficiency and functionality before running the experiments. Through the experiments, we demonstrate the use of `DeepGame` on the attention models and compare its outputs for different models.

Acknowledgements

I would like to thank my supervisor, Prof. Marta Kwiatkowska, for her patient guidance throughout the project, and for giving me access to the Lovelace server. I would also like to thank Emanuele La Malfa, Bill Shao, Benjie Wang and Denitsa Markova for their advice on the programming part of the project. Finally, I would like to express my gratitude towards Prof. Christoph Hasse, who gave me suggestions about how to improve my report writing.

Contents

Chapter 1

Introduction

1.1 Motivation

Machine learning has achieved significant successes in solving artificial intelligence problems in recent years [HKWT18]. Deep learning, a subfield of machine learning, has attained state-of-the-art performance in complex tasks such as computer vision and natural language processing [GBC16], using models known as artificial neural networks. The improvement in deep learning is largely attributed to greater dataset sizes and computation power, allowing much larger neural networks to learn more complex information in a given amount of time [GBC16]. Moreover, innovations in model architectures, such as the introduction of attention mechanisms to build attention models, have further enhanced the performance of neural networks [KSZQ20]. Inspired by the concept of attention in psychology, attention mechanisms in deep learning aim to disregard irrelevant features and concentrate computation power on more relevant ones. In image classification tasks, attention mechanisms can enhance the classification accuracy of neural networks [BZV$^+$19, WPLK18], and they allow the use of lower-resolution inputs without significantly compromising the performance [JSZ$^+$15].

The development in deep learning has sparked interest in industrial applications of neural networks, and some applications, like self-driving cars and medical diagnosis, are safety-

critical [GLB+18]. However, the large and complex models can represent highly non-linear and unstable mappings from the inputs to the outputs. Indeed, in the context of object recognition tasks, [SZS+13] found that by applying a slight perturbation to a correctly classified input image, we can cause a state-of-the-art neural network to misclassify the resulting image. We call the resulting image an "adversarial example", and we say a model is "robust" if it can continue to classify an input correctly despite such perturbations to the input. Considering such vulnerabilities, it is important to evaluate and improve the models' robustness towards adversarial examples, thereby providing better safety guarantees that are crucial to safety-critical applications.

There are various properties quantifying the robustness of a neural network, and different verification techniques exist to compute provable guarantees on the properties [HKR+20]. In this project, we focus on a pointwise robustness property called the "maximum safe radius" (MSR). Intuitively, the MSR of a model for a given input is the distance (with respect to a given metric) from the input to the closest adversarial example. We bound the MSR via the game-based verification algorithms proposed in [WWR+18]. The algorithms reduce the problem of finding the MSR to a search problem with a discrete state space. In the search problem, the objective is to find an adversarial example on a grid that is representative of the entire input space. Two players controlling different parts of the search cooperate to find a solution. This problem is intractable, and the algorithms can be interrupted to yield bounds on the solution, i.e., they are anytime algorithms. Monte Carlo tree search is the algorithm used to look for an upper bound on the MSR, and admissible A* search to look for a lower bound. [WWR+18] implemented the algorithms in a software tool named `DeepGame` and used the tool to conduct experiments on convolutional neural networks (CNNs) for image classification problems.

This project is based on [WWR+18], as we improve the efficiency of the tool, adapt it to our project, and assess its suitability for neural networks with attention mechanisms.

However, some of the theoretical results, such as the validity of the lower bounds, require a parameter that depends on the Lipschitz constant of the neural network. The Lipschitz constant bounds how fast the neural network's output changes with respect to changes in the input [CMN19]. There exists literature on finding good Lipschitz constants of feed-forward neural networks [FRH+19], but there is currently no such well-developed algorithm for neural networks with attention mechanisms. Hence, we make prior assumptions about some parameter values in our analysis.

1.2 Contributions

We categorize our contributions into two parts: programming and experimenting.

In the programming part,

- we adapt `DeepGame` from [WWR+18], which uses the tensorflow-gpu 1.4.0 library, to use tensorflow-gpu 2.4.1 instead, as it is easier to find well-maintained documentation and compatible libraries for the latter;
- we enhance the performance of `DeepGame` by parallelizing multiple runs of experiments, and by pruning away nodes with visited states in the A* search. There is empirical evidence that the optimizations improved the efficiency of the search algorithms;
- we add code to `DeepGame` to facilitate data collection, saving data such as the best bounds obtained, the number of iterations of the searches, and the difference between the closest adversarial example and the original input; we also write code to extract the information from the files containing the data, and collate the information for data analysis;
- we introduce two simple attention mechanisms inspired by [BZV+19] and [WPLK18] and implement them in Python;

- we build and train neural networks, with and without the attention mechanisms, on the German Traffic Sign Recognition Dataset (GTSRB)[1] from [SSSI11]. The resulting neural networks are the models we use in the experiments.

In the experimenting part,

- we use the improved `DeepGame` to evaluate the robustness of the models, demonstrating its use on attention models; we also compare the performance across different models, analyzing the best bounds obtained, the adversarial examples found and the features extracted;
- we then vary parameters in `DeepGame` to investigate how they influence the performance of the tool.

The improved `DeepGame` code is available at the anonymized repository, `https://anonymous.4open.science/r/DeepGame-Attention`. The changes are visible in its difference from the original code at `https://github.com/TrustAI/DeepGame/tree/6a3f0d1a48a66a3c1b48be33c09c84ca368e2f1a`.

We demonstrated that `DeepGame` can find adversarial examples for attention models effectively, and its performance on attention models is comparable to that on pure CNNs in terms of bounding the MSR. We also found evidence suggesting that the attention models we built are not necessarily more robust than pure CNNs at the image classification task.

1.3 Overview of the report

We now summarize the rest of this report.

- Chapter 2 gives an overview of the ideas behind the algorithms in `DeepGame`, and describes the two attention mechanisms we extract from existing papers;

[1] We obtained the dataset by email from the authors of [SSSI11].

- Chapter 3 describes the main improvements we made to `DeepGame`;
- Chapter 4 describes the design of our experiments, including the choices of parameters and variables;
- Chapter 5 presents and analyzes the results from the experiments;
- Chapter 6 concludes the report;
- Chapter 7 reflects upon the project and suggests areas that future research may explore.

Chapter 2

Related Work

2.1 Game-based verification of deep neural networks

This section provides an overview of the definitions, theorems and algorithms that are necessary for understanding how `DeepGame` bounds the minimum safe radius (MSR). The core contents and notations in this section are from [WWR$^+$18], while we provide additional descriptions and diagrams for easier comprehension. Further details and mathematical proofs are available in [WWR$^+$18].

2.1.1 Preliminary definitions

Consider a neural network N for a classification task. Let C be the set of all *classes*, and P_0 be the set of *dimensions* of an input so $|P_0|$ is the number of dimensions. For example, in an image classification task, the input α is an image with width w, height h and ch channels, so it has $|P_0| = w \cdot h \cdot ch$ dimensions, as shown in Figure 2.1. Moreover, we say that the image input has $w \cdot h$ pixels, each associated with ch dimensions.

For an input α and a class $c \in C$, let $N(\alpha, c)$ denote N's predicted probability that α belongs to c. Let $N(\alpha)$ be the class which N classifies α into. Let $\alpha[i]$ denote the value of the i-th dimension of α. Without loss of generality, we assume that each input dimension is

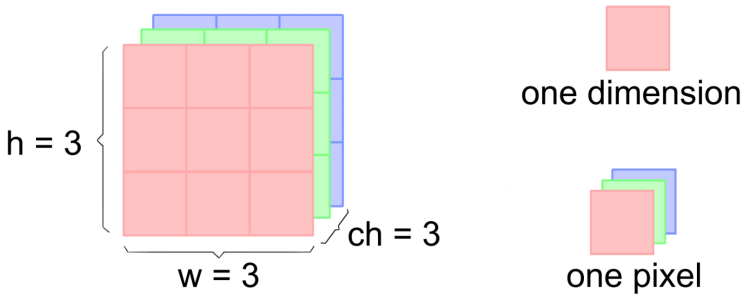

Figure 2.1: An illustration of a dimension and a pixel of an image input. The input has the shape (w, h, ch) where $w = h = ch = 3$.

normalized to the interval $[0, 1]$, so we can denote the input space by D where $D = [0, 1]^{|P_0|}$.

For a vector $\mathbf{v} = (v_1, ..., v_n)$, let $\|\mathbf{v}\|_k$ denote the length of $\mathbf{v}$ measured under the L_k distance function. That is, $\|\mathbf{v}\|_k = (\sum_{i=1}^{n} |v_i|^k)^{1/k}$. We consider $k \geq 1$.

We only consider *Lipschitz* networks, as defined below. From [RHK18, SZS+13], given that the inputs are bounded, most common types of neural network layers are Lipschitz continuous, including the layers in CNNs and our attention mechanisms. Hence, our models are Lipschitz networks.

Definition 1 *A neural network N is a Lipschitz network with respect to a distance function L_k if there exists a constant $\hbar_c > 0$ for every class $c \in C$, such that for all inputs $\alpha, \alpha' \in D$, we have*

$$|N(\alpha', c) - N(\alpha, c)| \leq \hbar_c \cdot \|\alpha' - \alpha\|_k \tag{2.1}$$

where $\hbar_c$ is a Lipschitz constant for the class c.

Equivalently, when $\alpha \neq \alpha'$, $\hbar_c \geq \frac{|N(\alpha', c) - N(\alpha, c)|}{\|\alpha' - \alpha\|_k}$, so $\hbar_c$ is like an upper bound on the rate of change of the confidence that the input belongs to c, with respect to the change of the input measured by L_k.

The "point" in the "pointwise robustness" refers to a fixed input α. It is therefore useful to define the neighbourhood of a given input.

Definition 2 *Given an input α, a distance function L_k, and a distance d, the d-neighbourhood of α with respect to L_k is*

$$\eta(\alpha, L_k, d) = \{\alpha' \mid \|\alpha' - \alpha\|_k \leq d\}. \tag{2.2}$$

Hence, $\eta(\alpha, L_k, d)$ is the set of inputs within the distance d of α, with the distance measured under L_k. Figure 2.2 demonstrates the concept for the special case where α has two dimensions and $k = 2$.

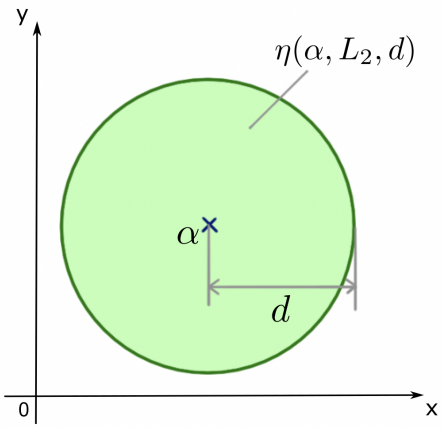

Figure 2.2: The green region shows the d-neighbourhood of α with respect to L_2, which we denote by $\eta(\alpha, L_2, d)$. The inputs have two dimensions, labelled x and y respectively.

We also introduce the notation d^ϵ to denote $d + \epsilon$ for any arbitrarily small $\epsilon > 0$, so d^ϵ is a number greater than d. This is used in our definition of the maximum safe radius problem.

Now we describe a systematic way of manipulating the given input, which is essential in our search for adversarial examples. Let $\tau > 0$ be a real number called the *manipulation*

magnitude, $X \subseteq P_0$ be a subset of dimensions, $\psi : P_0 \to \mathbb{Z}$ be an *instruction function*[1].

Definition 3 *Given $\tau > 0$, $X \subseteq P_0$, and $\psi : P_0 \to \mathbb{Z}$, the input manipulation operation $\delta_{\tau,X,\psi} : D \to D$ is defined by*

$$\delta_{\tau,X,\psi}(\alpha[i]) = \begin{cases} \alpha[i] + \psi(i) \cdot \tau, & \text{if } i \in X \\ \alpha[i], & \text{otherwise.} \end{cases} \tag{2.3}$$

where $\delta_{\tau,X,\psi}(\alpha[i])$ means the value of the i-th dimension of $\delta_{\tau,X,\psi}(\alpha)$. For bounded domains such as $D = [0,1]^{|P_0|}$, we restrict the manipulation result to be within the domain.

We have ψ mapping a dimension to the number of times we manipulate the dimension by τ or $-\tau$, and X restricting the dimensions in which the manipulations occur. The manipulation in each dimension by $\delta_{\tau,X,\psi}$ is hence a multiple of τ. This gives rise to the concept of *atomic manipulations*.

Definition 4 *Given $\tau > 0$, $X \subseteq P_0$, define $\Delta(X)$ as the set of* atomic *input manipulations, each of the form δ_{τ,X_1,ψ_1}, such that $X_1 \subseteq X$, $|X_1| = 1$ and $\psi_1(i) \in \{-1, +1\}$ for all dimensions i.*

Each input manipulation in $\Delta(X)$ is *atomic* because it manipulates a single dimension by τ in either direction. We can implement any input manipulation $\delta_{\tau,X,\psi}$ by a sequence of (possibly repeated) atomic manipulations from $\Delta(X)$, and we use atomic manipulations in our search algorithms. The choice of X is related to the idea of features, which we will define next.

Given an input $\alpha \in D$, we associate a *feature* with a set of dimensions. We use feature extraction methods such that the set of all features corresponds to a partition of the set of all dimensions P_0, and the result of the feature extraction may depend on the input. In the

[1] In [WWR+18], the codomain of ψ is $\mathbb{N}$ instead of $\mathbb{Z}$, but we believe this is an error, as the subsequent definition of atomic input manipulations has the instruction function mapping a dimension to a negative number.

game-based algorithms, one player's moves involve choosing features, while the other player chooses atomic manipulations constrained to the chosen features.

Definition 5 *Let λ denote a feature, and $P_\lambda \subseteq P_0$ the dimensions represented by λ for a given input α. A feature extraction function Λ maps an input α to a set of features, such that $\{P_\lambda \mid \lambda \in \Lambda(\alpha)\}$ is a partition of P_0.*

For the discussion of the algorithms, it is convenient to consider adversarial examples within a given distance of the fixed input.

Definition 6 *Given an input $\alpha \in D$, a distance function L_k, a distance d, and a class $c \in C$ such that $c \neq N(\alpha)$, we define*

$$adv_{k,d}(\alpha, c) = \{\alpha' \mid \alpha' \in \eta(\alpha, L_k, d) \wedge N(\alpha') = c \neq N(\alpha)\}, \tag{2.4}$$

$$adv_{k,d}(\alpha) = \bigcup_{c \in C, c \neq N(\alpha)} adv_{k,d}(\alpha, c). \tag{2.5}$$

Equivalently, $adv_{k,d}(\alpha, c)$ is the set of adversarial examples within distance d from α classified into c, while $adv_{k,d}(\alpha)$ does not target a specific class c. The results below are for $adv_{k,d}(\alpha, c)$, but similar results hold for $adv_{k,d}(\alpha)$.

2.1.2 Maximum safe radius problem

We now define the maximum safe radius (MSR) problem and the finite maximum safe radius (FMSR) problem, and state the relationships between their solutions.

Definition 7 *Given a distance function L_k, a distance d, an original input $\alpha \in D$, and a class $c \neq N(\alpha)$, the maximum safe radius (MSR) problem is the problem of finding the MSR, as defined by*

$$\texttt{MSR}(k, d, \alpha, c) = \min_{\alpha' \in D} \{\|\alpha - \alpha'\|_k \mid \alpha' \in adv_{k,d}(\alpha, c)\} \tag{2.6}$$

with $\texttt{MSR}(k, d, \alpha, c) = d^\epsilon$ if $adv_{k,d}(\alpha, c) = \emptyset$.

Hence, $\texttt{MSR}(k, d, \alpha, c)$ is the minimum distance from α to an adversarial example within distance d, or d^ϵ if no such adversarial example exists. There is no adversarial example at a distance less than $\texttt{MSR}(k, d, \alpha, c)$ from α, so $\texttt{MSR}(k, d, \alpha, c)$ is like an absolute safe radius from α, as shown in Figure 2.3.

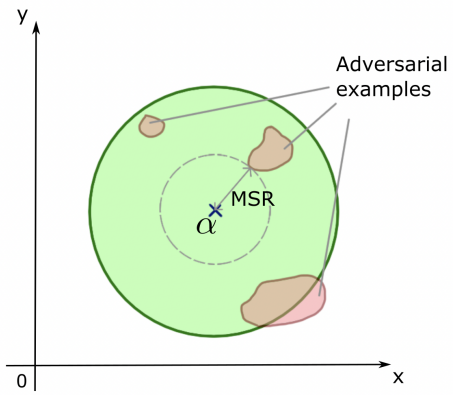

Figure 2.3: A figure showing the maximum safe radius (MSR) of an input α. The green region is $\eta(\alpha, L_2, d)$, and the red regions represent sets of adversarial examples. The inputs have two dimensions, labelled x and y respectively.

Since the input space D can have an infinite cardinality, and we hope to use finite optimization algorithms to approximate MSR with provable guarantees, we discretize the MSR problem.

Definition 8 *Given a manipulation magnitude $\tau \in (0, 1]$ and other parameters as in the definition of the MSR problem, let Ψ denote the set of all possible instruction functions. The finite maximum safe radius (FMSR) problem is to find*

$$\texttt{FMSR}(\tau, k, d, \alpha, c) = \min_{\Lambda' \subseteq \Lambda(\alpha)} \min_{X \subseteq \bigcup_{\lambda \in \Lambda'} P_\lambda} \min_{\psi \in \Psi} \{\|\alpha - \delta_{\tau,X,\psi}(\alpha)\|_k \mid \delta_{\tau,X,\psi}(\alpha) \in adv_{k,d}(\alpha, c)\} \quad (2.7)$$

and we let $\texttt{FMSR}(\tau, k, d, \alpha, c) = d^\epsilon$ if $adv_{k,d}(\alpha, c) = \emptyset$.

Intuitively, the FMSR problem replaces the continuous variable $\alpha' \in D$ with the discrete variables Λ', X and ψ, choosing a set of features, and then a set of dimensions from the chosen

features and an input manipulation in those dimensions to obtain an adversarial example. As a result, it considers only inputs that are obtained from α via manipulations by multiples of τ.

Since the closest adversarial example solving the FMSR problem is a candidate solution to the MSR problem, it follows that $\texttt{MSR}(k, d, \alpha, c) \leq \texttt{FMSR}(\tau, k, d, \alpha, c)$. To derive a lower bound on $\texttt{MSR}(k, d, \alpha, c)$, however, we need to fulfil more conditions.

Definition 9 *An input $\alpha' \in \eta(\alpha, L_k, d)$ is called a* τ-grid input *if for all dimensions $i \in P_0$, there exists an integer $n \geq 0$ such that $|\alpha[i] - \alpha'[i]| = n \cdot \tau$. Let $G(\alpha, k, d)$ be the set of τ-grid inputs in $\eta(\alpha, L_k, d)$.*

The set of τ-grid inputs is the set of inputs searched in the FMSR problem, as shown in Figure 2.4.

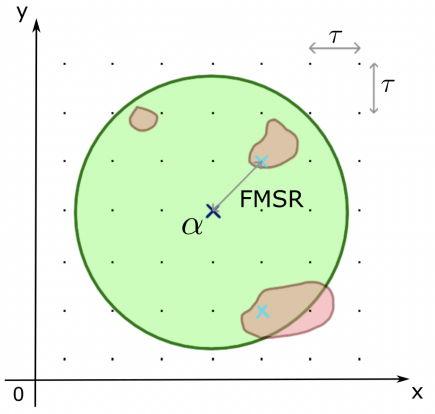

Figure 2.4: A figure showing the finite maximum safe radius (FMSR) of an input α, with the manipulation magnitude τ. The green region is $\eta(\alpha, L_2, d)$, and the red regions represent sets of adversarial examples. The grey dots show the τ-grid and the blue "x"s show the adversarial examples considered in the FMSR problem. The inputs have two dimensions, labelled x and y respectively.

Definition 10 *Suppose we are considering adversarial examples of α. An input $\alpha_1 \in \eta(\alpha, L_k, d)$ is a* misclassification aggregator *with respect to some $\beta > 0$ if, for all inputs $\alpha_2 \in \eta(\alpha_1, L_k, \beta)$, $N(\alpha_2) \neq N(\alpha)$ implies $N(\alpha_1) \neq N(\alpha)$.*

Equivalently, fixing a positive distance β, we call α_1 a misclassification aggregator if the correct classification of α_1 implies the correct classification of any input within distance β of α_1. By "correct classification", we mean the classification of the input into $N(\alpha)$.

Lemma 1 *Let* $D(k,\tau) = \sqrt[k]{|P_0| \cdot \tau^k}$. *Then* $\eta(\alpha, L_k, d) \subseteq \bigcup_{\alpha' \in G(\alpha,k,d)} \eta(\alpha', L_k, \frac{1}{2}D(k,\tau))$.

This lemma states that by taking the union of the $\frac{1}{2}D(k,\tau)$-neighbourhoods of all τ-grid inputs in $\eta(\alpha, L_k, d)$, we will include all inputs in $\eta(\alpha, L_k, d)$. Figure 2.5 illustrates the lemma in the special case where $k = 2$ and the inputs have two dimensions.

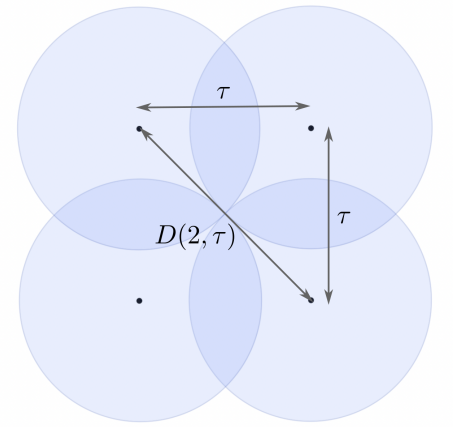

Figure 2.5: A figure showing how the $\frac{1}{2}D(2,\tau)$-neighbourhoods of τ-grid points, represented by the blue discs, can cover the entire 2-dimensional input space.

Lemma 2 *Under the condition that all* τ*-grid inputs are misclassification aggregators with respect to* $\frac{1}{2}D(k,\tau)$*, we have* $\texttt{MSR}(k,d,\alpha,c) \geq \texttt{FMSR}(\tau,k,d,\alpha,c) - \frac{1}{2}D(k,\tau)$.

For an intuitive explanation of this lemma, assume for contradiction that $\texttt{MSR}(k,d,\alpha,c)$ is strictly smaller than $\texttt{FMSR}(\tau,k,d,\alpha,c) - \frac{1}{2}D(k,\tau)$. Let α' be the adversarial example associated with $\texttt{MSR}(k,d,\alpha,c)$. By Lemma 1, the $\frac{1}{2}D(k,\tau)$-neighbourhoods of τ-grid inputs covers all possible inputs in $\eta(\alpha, L_k, d)$, so α' is in the $\frac{1}{2}D(k,\tau)$-neighbourhood of some τ-grid input α''. By our assumption, the distance between α'' and α is strictly smaller than $\texttt{FMSR}(\tau,k,d,\alpha,c)$. However, by the condition that all τ-grid inputs are misclassification aggregators, α'' is an

adversarial example at a distance smaller than $\texttt{FMSR}(\tau, k, d, \alpha, c)$ from α, which is a contradiction.

For the lower bound in Lemma 2 to hold, we just need to ensure that all τ-grid inputs are misclassification aggregators. Given the neural network N, the input α, the function L_k, and the distance d for the MSR and the FMSR problems, we can achieve the condition by making τ small enough such that for all $c \in C$, for all $\alpha' \in G(\alpha, k, d)$,

$$D(k, \tau) \leq \frac{2g(\alpha', N(\alpha'))}{\max_{c \in C, c \neq N(\alpha')}(\hbar_{N(\alpha')} + \hbar_c)} \tag{2.8}$$

where $g(\alpha', c) = \min_{c' \in C, c' \neq c}\{N(\alpha', c) - N(\alpha', c')\}$. This method uses Lipschitz constants of N to bound the rate of change of N's outputs, and makes τ small enough so that the changes in the outputs are insufficient to alter the classification near τ-grid inputs.

Therefore, given that τ satisfies Equation (2.8), we have $\texttt{FMSR}(\tau, k, d, \alpha, c) - \frac{1}{2}D(k, \tau) \leq \texttt{MSR}(k, d, \alpha, c) \leq \texttt{FMSR}(\tau, k, d, \alpha, c)$. Theoretically, the search algorithms we use for bounding $\texttt{FMSR}(\tau, k, d, \alpha, c)$ converge to the true value eventually, so we have the theoretical guarantee that we can eventually compute both the upper and the lower bounds on $\texttt{MSR}(k, d, \alpha, c)$ proven above. Nevertheless, the finite optimization problem is still intractable, as the number of dimensions $|P_0|$ of the input is usually large and the state space $G(\alpha, k, d)$ has size exponential in $|P_0|$. In practice, we obtain the best bounds found under some time constraint, and that is why the performance of the algorithms on different models is of interest.

2.1.3 Searching the game tree

Although [WWR+18] discusses algorithms bounding $\texttt{FMSR}(\tau, k, d, \alpha, c)$, the implementation in `DeepGame` does not target any specific class c. Hence, we introduce the game and the

algorithms for $\texttt{FMSR}(\tau, k, d, \alpha)$ instead, where

$$\texttt{FMSR}(\tau, k, d, \alpha) = \min_{c \in C} \texttt{FMSR}(\tau, k, d, \alpha, c). \tag{2.9}$$

Let the two players be Player `I` and Player `II`. In the search, the two players alternate their actions. Player `I` starts the game by choosing a feature, Player `II` then chooses an atomic manipulation restricted to the chosen feature, and they repeat the two steps until they reach a terminal node. The root node contains the initial input α. A node expansion creates one child node for each valid action at the current node. A path from the root node corresponds to the sequence of atomic manipulations along the path, and the node we reach by traversing the path corresponds to the input obtained by the manipulations from α. A terminal node corresponds to an input that is either in $adv_{k,d}(\alpha)$ or outside $\eta(\alpha, L_k, d)$, i.e., either an adversarial example or an input too far from α.

Monte Carlo tree search

For an upper bound on $\texttt{FMSR}(\tau, k, d, \alpha)$, we use Monte Carlo tree search (MCTS), as shown in Algorithm 1.

The *termination condition* could be a time constraint, a user interrupt, or the completion of the search (which is unlikely in practice). The random *simulation* of moves in Line 7 does not add any nodes to the search tree T, and we can perform a few independent simulations and take the minimum *cost*. The exploration-exploitation ratio R determinines the trade-off between exploration and exploitation. Specifically, to *select* a leaf node for expansion, we traverse the current search tree from the root down to a leaf by a stochastic process, sampling the next node we traverse to using the probability distribution defined below. Given a node o, let n_o denote the number of times we back-propagated costs to o, and r_o the sum of all costs back-propagated to o. Let o be an internal node with the set C_o of child nodes. For

Algorithm 1 Monte Carlo tree search for an upper bound

Hyperparameters: exploration-exploitation ratio R

Input: neural network N, input α, distance function L_k, distance d, manipulation magnitude τ, feature extraction function Λ, termination condition tc

Output: an upper bound on $\texttt{FMSR}(\tau, k, d, \alpha)$ of N

Initialise the search tree T to contain a single node, with input α and cost 0;
$bestUpperBound \leftarrow d^{\epsilon}$
while $\neg tc$ **do**
    Select a leaf node o of T (with R influencing the selection);
    Expand o and add its child nodes to T;
    for each new node o' created in the expansion **do**
        Randomly *simulate* moves from o' until a terminal node o'';
        $\alpha'' \leftarrow$ the input associated with o'';
        $cost \leftarrow \|\alpha'' - \alpha\|_k$;
        if $cost < bestUpperBound$ **then**
            $bestUpperBound \leftarrow cost$
        end if
        Backpropagate cost up to the root node;
    end for
end while
return $bestUpperBound$

each $o' \in C_o$, define its weight

$$w(o, o') = \frac{d \cdot n_{o'}}{r_{o'}} + R\sqrt{\frac{\ln n_o}{n_{o'}}} \tag{2.10}$$

and if o is the node we have traversed to, the probability of choosing o' as the next node is

$$Prob_o(o') = \frac{w(o, o')}{\sum_{o'' \in C_o} w(o, o'')} \tag{2.11}$$

which is higher for child nodes with greater weights. Note that the first term of the weight is inversely proportional to the average cost $\frac{r_{o'}}{n_{o'}}$ so it serves as the average reward of o', and the second term is greater if the number of back-propagations via o' to o is small compared to that via other child nodes, i.e., if o' is less explored. Hence, the greater R is, the more we favour the exploration of a less explored node over the exploitation of a currently good node.

Admissible A* search

We use A* search with an admissible heuristic for a lower bound on $\texttt{FMSR}(\tau, k, d, \alpha)$, as shown in Algorithm 2. Using the same the definition of $g(\alpha', N(\alpha'))$ as that in Equation (2.8), we can construct an admissible heuristic function.

Lemma 3 *Let o be a node in the search tree, and α' be the input associated to o. Then we can consider α' as the state of o. Define the heuristic function h by*

$$h(\alpha') = \frac{g(\alpha', N(\alpha'))}{\max_{c \in C, c \neq N(\alpha')}(\hbar_{N(\alpha')} + \hbar_c)}. \tag{2.12}$$

We have that h is admissible. (Note that h depends only on the state of a node.)

In the implementation, `DeepGame` uses a lower bound of this heuristic computed from $g(\alpha', N(\alpha'))$ and τ, which is still admissible.

Since only nodes after Player `II`'s action have inputs as their states, we combine each pair of actions by both players, so that the evaluation function v determines the choice of the next feature and the next atomic manipulation, i.e., the next atomic manipulation chosen over all possible dimensions. An expansion will hence create a child node resulting from every possible atomic manipulation from the current input. By the optimality property of admissible A* search, if it expands a node with distance d' from α without encountering an adversarial example, then there is no adversarial example with distance less than d' from α. Hence, an adversarial example found by the algorithm must be a closest adversarial example on the τ-grid.

2.2 Attention models

This section describes the two attention mechanisms we use to construct attention models. We assume the reader's familiarity with convolutional neural networks (CNNs), as described

Algorithm 2 Admissible A* search for a lower bound

Input: neural network N, input α, distance function L_k, distance d, manipulation magnitude τ, feature extraction function Λ, termination condition tc

Output: a lower bound on $\texttt{FMSR}(\tau, k, d, \alpha)$ of N

Initialise $frontier$ to contain a single node o whose state is α;
Let v be the evaluation function, so $v(o) \leftarrow h(\alpha)$;
$bestLowerBound \leftarrow 0$;
while $\neg tc$ **do**
    Let o' be a node in $frontier$ with the minimum value under v;
    Let α' be the state of o';
    $cost \leftarrow \|\alpha' - \alpha\|_k$;
    if $cost > d$ **then return** d^ϵ;
    else if α' is an adversarial example **then return** $cost$;
    end if
    if $cost > bestLowerBound$ **then**
        $bestLowerBound \leftarrow cost$;
    end if
    Remove o' from $frontier$ and expand o', adding all its child nodes to $frontier$;
    for each child node o'' of o' **do**
        Let α'' be the state of o'';
        $v(o'') \leftarrow \|\alpha'' - \alpha\|_k + h(\alpha'')$;
    end for
end while
return $bestLowerBound$;

in [GBC16]. We refer to the output of any convolutional layer or pooling layer as a "feature map", with a shape of the form (H, W, F_{in}), so the feature map has $H \cdot W$ *pixels* and F_{in} *channels*. The papers we refer to describe attention mechanisms with richer structures, and we extract a very specific kind of attention mechanism from each paper for this project. The mechanisms can be added to any part of a CNN that produces a feature map.

2.2.1 Augmenting CNNs using self-attention

We extract one of the attention mechanisms, called self-attention, from the [BZV+19]. As illustrated in Figure 2.6, given a feature map with the shape (H, W, F_{in}), we flatten it to a matrix X with the shape $(H \cdot W, F_{in})$. We can interpret a row of X as the representation of a pixel by the F_{in} channels. We then use linear transformations to obtain the matrices XW_q,

XW_k, and XW_v respectively. The matrices W_q, W_k and W_v are all trainable parameters. Let (F_{in}, d_k) be the shape of W_q and W_k, and (F_{in}, d_v) be the shape of W_v. Each row of XW_q is like the *query vector* of a pixel, each row of XW_k the *key vector*, and each row of W_v the *value vector*.

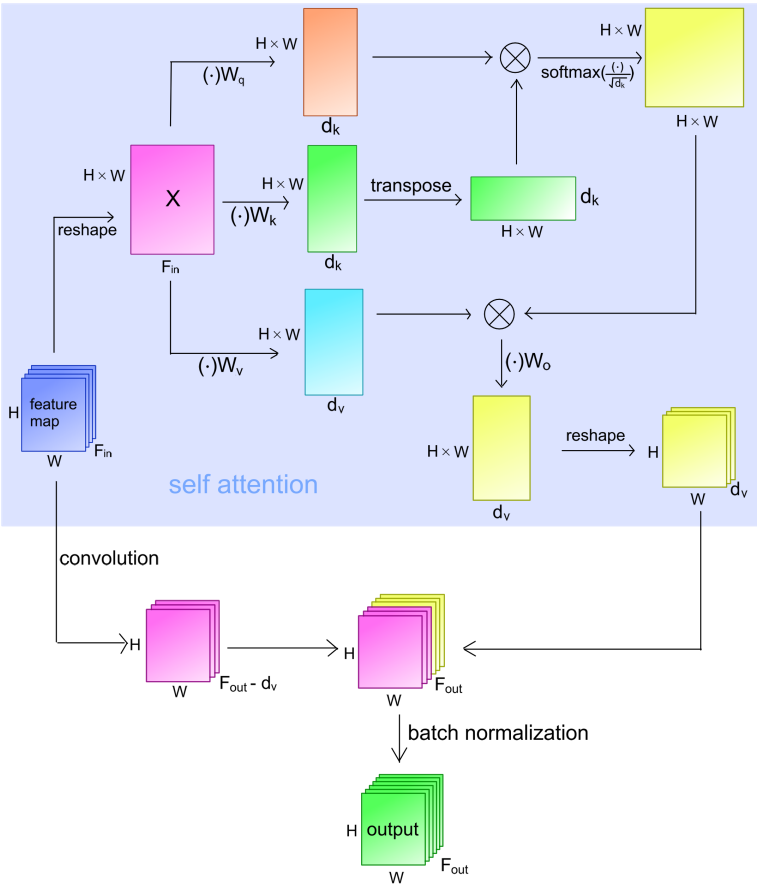

Figure 2.6: A segment of the neural network architecture that uses a self-attention module. The symbol $\otimes$ denotes matrix multiplication, and the dimensions of the two input matrices and the output matrix indicate which input matrix is on the left of the multiplication and which is on the right.

Consider a pixel corresponding to a row of XW_q. Multiplying the row with $(XW_k)^T$ has the effect of taking the dot product between the query vector and every key vector so that after scaling the result and applying the softmax function, we have a row vector that acts

like weights for the next step. In the next step, we take the weighted sum of all value vectors followed by another linear transformation. The resulting row vector is a new representation of the pixel. We then reshape the new matrix to match the shape of a feature map. Using the idea of "augmentation" from [BZV+19], we also perform a convolutional operation on the original feature map and concatenate this output with the output from the self-attention module. We can tune the hyperparameter d_v to control the proportion that the attention module's output takes in the concatenated output.

2.2.2 Spatial attention in Convolutional Block Attention Module (CBAM)

From [WPLK18], we extract the spatial attention module, which is the other attention mechanism we use. Figure 2.7 illustrates its architecture. Given a feature map of the shape (H, W, F_{in}), we perform both max pooling and average pooling over the channel dimension, concatenate the pooling results, and pass it through a convolutional layer. After an element-wise application of the sigmoid function, we obtain weights for the $H \cdot W$ pixels. We then broadcast the weights to all channels, do an element-wise multiplication with the original feature map and then an element-wise addition with the original feature map to give the output.

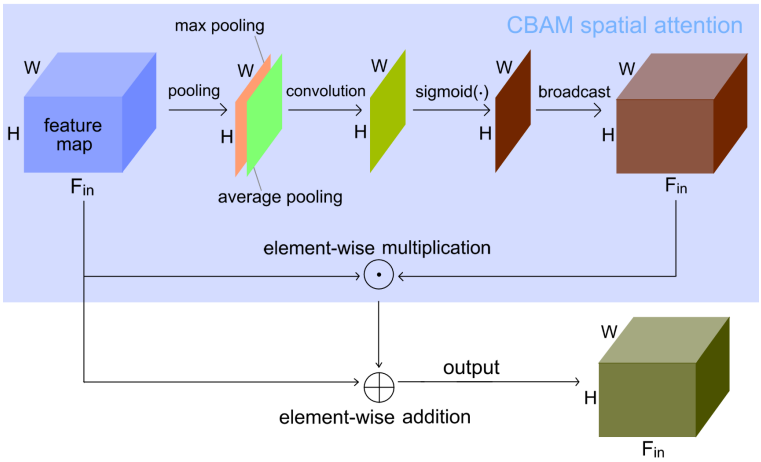

Figure 2.7: A segment of the neural network architecture that uses a CBAM-spatial-attention module.

Chapter 3

Improvement on DeepGame

This chapter describes the main changes we made to `DeepGame` for functionality and efficiency. More details can be found in the Anonymous Github repository.

3.1 Adaptation to this project

We updated the code to use TensorFlow 2 instead of TensorFlow 1. Since Keras is integrated into TensorFlow 2, there are changes in the libraries that we can use.

We also created modules for building and training the different types of models. Using the existing DataCollection module, we added code to save data in files, as well as code to extract, preprocess and analyze the data from the files.

Moreover, we fixed some bugs in the program. For example, the original implementation of the MCTS algorithm avoids creating a child node by an atomic manipulation that only causes negligible changes in the model's outputs. We may view this as a heuristic for pruning away less promising parts of the search tree, which is justifiable since the state space is intractable. However, in some of our experiments, especially on the attention models, a node chosen for expansion has all its child nodes pruned, which caused an error as the program

tries to randomly sample a node in an empty set. Continuing to use this pruning method, we fixed the problem by removing the node whose child nodes are all pruned and repeating the tree traversal to select another leaf node for expansion. If we traverse to a previously expanded node, i.e., an internal node of the search tree, and find all its children removed this way, we terminate the search with an appropriate description in the log. Such a situation would indicate that our heuristic guiding the MCTS algorithm, which is determined by the exploration-exploitation ratio, is not performing well in finding the best search direction.

3.2 Optimization for efficiency

Our major optimization was the use of concurrent programming for running multiple experiments in parallel. Theoretically, different runs of experiments are independent, so we do not need any synchronization. However, in a preliminary experiment where we used bash commands to run multiple experiments in parallel, we encountered an error due to the different processes trying to access the same file. This was unavoidable as they share the files containing the models and the datasets. Hence, we used the Python Multiprocessing library to make the file accesses mutually exclusive, but allow the rest of the experiments to run in parallel. Since the experiments can run for hours, the overhead is negligible when the number of processes is fewer than the number of cores in the computer.

We also optimized the A* search algorithm. Recall from Algorithm 2 that the *cost* of a node is the distance from the node's state α' to the initial state α. Hence, if we reach the same state α' in two distinct nodes, the two nodes have the same cost, and the searches from the two nodes are identical in terms of the states reached and their estimated costs. Hence, we can safely remove one of the two nodes, searching only once from the state α', i.e., we can use a graph search. The original program avoids a manipulation that undoes a previous manipulation, but it allows an input to be reached from different directions. For instance, let (i, n) denote n atomic manipulations in the dimension i from the initial state α. A set

of such tuples determines the resulting state. Consider α' obtained by the manipulations $\{((0,0),1),((0,1),1)\}$, α_1 by $\{((0,0),1)\}$, and α_2 by $\{((0,1),1)\}$. The original program expands both the node for α_1 and the node for α_2 to produce nodes for α', and the nodes for α' are then expanded separately. This causes the number of nodes with the same state to be exponential in the number of manipulated dimensions. We improved the efficiency by keeping a collection of the states of previously expanded nodes and pruning away any newly discovered state that has been discovered by a previous expansion. To represent a state concisely using discrete values, instead of recording a state $\alpha' \in D = [0,1]$ directly, we represent it by a collection of manipulations as we have demonstrated in the example above. Tests showed that the improved algorithm finds strictly tighter lower bounds for multiple randomly sampled inputs under the same 30-minute time constraint.

Chapter 4

Experiment Design

4.1 Building and training models

We follow the architectures of the CNNs in [WWR+18] to build the corresponding attention models and train the models on the German Traffic Sign Recognition Dataset (GTSRB). To check that the code for the attention models generates the intended model architectures, we use the `plot_model` function from the TensorFlow library to plot the generated architectures and ensure they agree with our expectation. Details about the architectures and the training are in the appendix. For brevity, we will call the pure CNNs "pure models", and call the other two types of models "self-attention models" and "CBAM-spatial-attention models". This section describes the preliminary experiments we conducted to determine some hyperparameters in the model architectures.

4.1.1 Proportion of attention in self-attention model

As described in [BZV+19], in each attention-augmented part of the self-attention model, we can tune the proportion of channels that are from the attention module. We tried the proportions of 0.25, 0.5, 0.75 and 1, and found that 0.25 gave the best validation accuracy, which is consistent with the result in [BZV+19]. Moreover, the proportion of 1 gave signif-

icantly lower accuracy, which suggests that the self-attention mechanism cannot replace the CNN architecture, and that is consistent with the motivation of using attention mechanisms as an auxiliary tool. Hence, we used 0.25 as the proportion in the subsequent experiments.

4.1.2 Downscaling inputs

We also trained all three types of models on the GTSRB images downscaled by different factors. Without downscaling, each input from the GTSRB dataset has the shape $(48, 48, 3)$, which gives $48 * 48 * 3 = 6912$ dimensions. The attention models converged to slightly lower validation accuracies than the pure model, and MCTS could hardly reach the third layer of the search tree after running for an hour due to the high dimensionality. After downscaling the inputs to the shape $(24, 24, 3)$, the attention models had almost the same validation accuracies as before, but the accuracies are now slightly higher than the pure model's accuracy. The lower dimensionality also led to exploration deeper down the search tree when we ran `DeepGame` on the models. Further downscaling to $(12, 12, 3)$ significantly reduced the accuracies, likely due to the loss of too much information in the inputs. Hence, to fully exploit the benefits of attention models and search deeper down the tree, we chose the shape $(24, 24, 3)$.

4.2 Feature extraction

We used the saliency-guided image segmentation procedure as our main feature extraction method, but we also extracted features using a method based on the Scale Invariant Feature Transform (SIFT) [Low04] for comparison. Both methods are used in [WWR+18], and the paper refers to the saliency-guided method as "grey-box" feature extraction since it depends on the model, and the other method as "black-box" since it is model-independent. For the grey-box method, the number of features $n_{features}$ is a hyperparameter. The method ranks pixels by their influence on the model's output and partitions them in that order into $n_{features}$ sets of about the same size. By partitioning a pixel (x, y) into a feature λ, we mean that the dimension (x, y, ch) for each channel ch is in P_λ. In contrast, the black-box method

determines the number of features, which varies depending on the model and the input. It identifies keypoints with invariant properties, and each keypoint corresponds to a feature. Every pixel is then assigned to a feature based on its distances from the keypoints and the keypoints' strengths.

4.3 Sampling inputs

Since robustness is only desirable provided that the input is correctly classified, i.e., the classification outcome of the model is the same as the ground truth, we only consider inputs correctly classified by all three models for robustness comparison. Due to resource limitations, we are not able to run the experiment on a large number of inputs to reach a statistical conclusion. Instead, we sample one input from each class uniformly at random and run the experiments on all models for the sampled inputs. Since there are 43 classes in the GTSRB dataset, we sampled 43 inputs.

4.4 Variables

Our objective is to compare the performance of `DeepGame` on the three types of models. We start by using the hyperparameter values in the original `DeepGame` code as default values. Subsequently, we consider $n_{features}$, R and the feature extraction method as variables that can potentially influence the performance, and vary them to observe the outcomes. Following the experiment design in [Wu20], we try the values 2, 4, 6 and 8 for $n_{features}$ which are below the default value 10. For the values of R, we try the values 0.5, 1.0, 2.0 and 4.0 around the default $\sqrt{2}$. Since the variables do not affect the A* search algorithm, as the algorithm is independent of the features extracted, we only study their effects on the performance of the MCTS algorithm.

4.5 Measurements

We observe a few aspects of the performance, namely the bounds obtained as the experiments progress, the adversarial examples found by MCTS, and the features extracted and manipulated.

Due to the stochastic nature of the MCTS algorithm and the randomness in the parallel processing, we choose to compare the bounds obtained by different models after the same number of node expansions (instead of the same running time). This approach is also used in [Wu20]. We will refer to a node expansion in the search algorithms as an "iteration". When comparing experiments that terminated at different numbers of iterations, we use the smallest one and only consider the progress of each experiment up to that number of iterations.

We are interested in which model obtains the best bound at the end of the experiment, and the fraction of iterations in which a model has the best bound. For practical applications, the adversarial examples can be useful in guiding the training of a robust model. Hence, the number of adversarial examples found is also of interest. Finally, the features provide insight into the robustness of our models at the specific inputs.

Chapter 5

Results and Discussion

In this chapter, we will describe and discuss the observations about the bounds, the features and the adversarial examples found in the experiments.

5.1 Bounds and convergence trends

This section describes the observations about bounds, highlighting similar patterns found across different sets of experiments. When we discuss upper bounds on the FMSR, the "best" bound refers to the least upper bound, which corresponds to the distance to the closest adversarial example found so far. Similarly, the "best" lower bound is the greatest lower bound. For convenience, we refer to the best bound obtained at the end of an experiment as the "best final bound", and we refer to "having the best bound in the most iterations" as "most consistently best bounded". In our first set of experiments using the default parameters, the feature extraction method is grey-box, the number of features $n_{features}$ is 10, the exploration-exploitation ratio R is $\sqrt{2}$, we perform three simulations for each newly discovered node in MCTS, and we consider the $\eta(\alpha, L_2, 10)$-neighbourhood of the original input α. We use $\tau = 1$ for the MCTS algorithm, and $\tau = 0.01$ for the A* search algorithm. $\tau = 1$ allows faster progress in MCTS, and the upper bounds are distances to adversarial examples found in the algorithm, so they are always valid. However, the validity of the lower bound is under the

assumption that our choice of $\tau = 0.01$ is small enough. This is an assumption we had to make due to the absence of tools that can bound the Lipschitz constants of attention models.

Using the parameters in the first set of experiments as a baseline, we maintained $R = \sqrt{2}$ and the grey-box feature extraction method as we altered $n_{features}$. Similarly, we maintained $n_{features} = 10$ and the grey-box method as we varied R, and we kept $n_{features} = 10$ and $R = \sqrt{2}$ when we tried the different feature extraction methods. We investigated how changes in $n_{features}$, R and the feature extraction method influence the fraction of samples where each model obtained the best bounds among all three models. We also fixed the model and observed the fraction of input samples where each parameter value obtained the best bounds among all values we tried for the parameter.

5.1.1 Upper bounds

None of the models had a clear advantage in finding the best final bound for all input samples. Instead, the model that obtained the best final bound depended largely on the inputs. In the first set of experiments, the pure model obtained the best final bound for about 21.95% of the input samples, the self-attention model 48.78%, and the CBAM-spatial-attention model 29.27%. Hence, the self-attention model obtained the best final bound for a significantly larger fraction of input samples than each of the other models, although the fraction is not a vast majority. The observations are similar for the other sets of experiments, as shown in Figure 5.1 and Figure 5.2, where we only see slight fluctuations of fractions as the variables change. Moreover, the input samples at which each model obtained the best final bound remained largely unchanged as we varied $n_{features}$, R and the feature extraction methods. We highlight that the MCTS algorithm used to obtain the upper bounds uses randomness, so the statistics are stochastic in nature.

There were similar patterns for the most consistently best bounded model. In the first set of experiments, the pure model is most consistently best bounded for 24.39% of the input

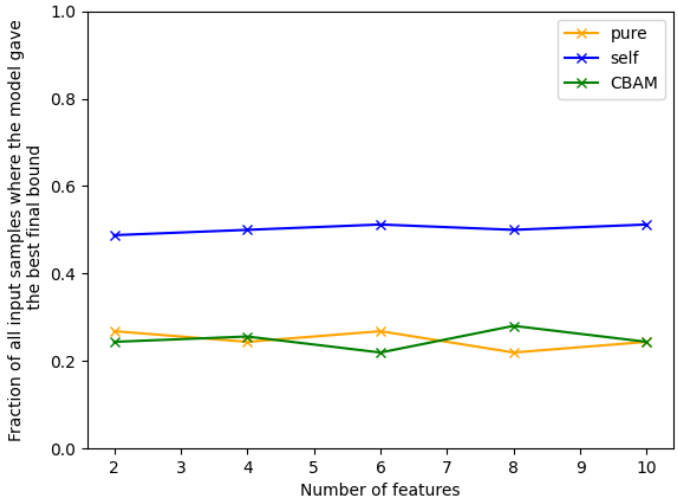

Figure 5.1: Graph showing the change in the relative performance of the models in terms of the *best final bound* as we varied $n_{features}$. For each $n_{features}$ value we tested, we plot the fraction of samples for which a model obtained the best final bound among all three models.

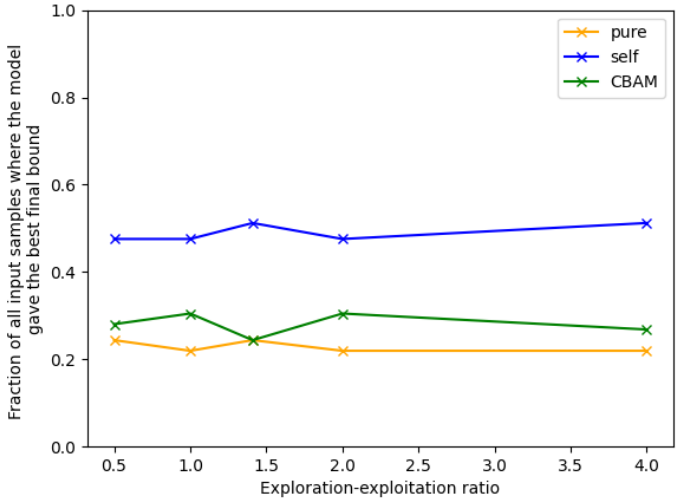

Figure 5.2: Graph showing the change in the relative performance of the models in terms of the *best final bound* as we varied R. For each R value we tested, we plot the fraction of samples for which a model obtained the best final bound among all three models.

samples, the self-attention model 51.22%, and the CBAM-spatial-attention model 24.39%. Varying $n_{features}$ and R does not change the fractions significantly, as shown in Figure 5.3 and Figure 5.4.

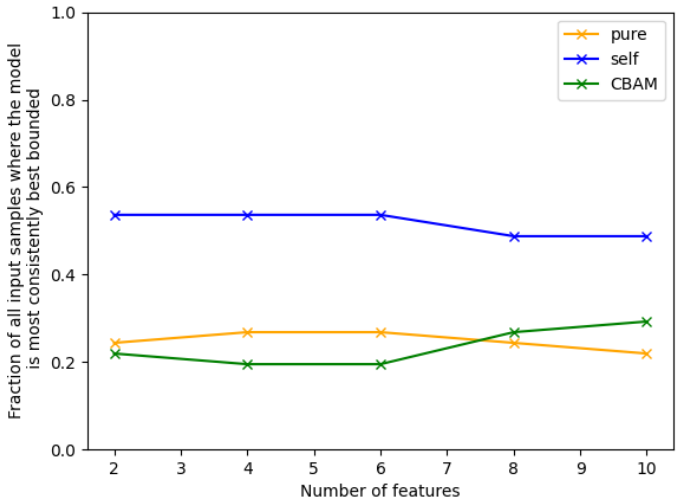

Figure 5.3: Graph showing the change in the *most consistently best bounded* model as we varied $n_{features}$. For each $n_{features}$ value we tested, we plot the fraction of samples for which a model was most consistently best bounded among all three models.

The input samples at which a model obtained the best final bound largely overlapped with the input samples at which the model is most consistently best bounded. This is likely because, for most input samples, MCTS finds bounds very close to the best final bounds in early iterations, so the order is the same for most iterations, as shown in Figure 5.5. Nevertheless, due to the stochastic nature of MCTS, there is no guarantee that the bounds have converged to the true FMSR.

Now we fix the model, and consider the $n_{features}$ value that gave the best final bound out of all the values we tested. Figure 5.6 shows that $n_{features} = 2$ gave the best final bound for the largest fraction of samples. Moreover, we observe similar trends in the pie charts for both attention models, where the fraction decreases and then increases as $n_{features}$ increases, and

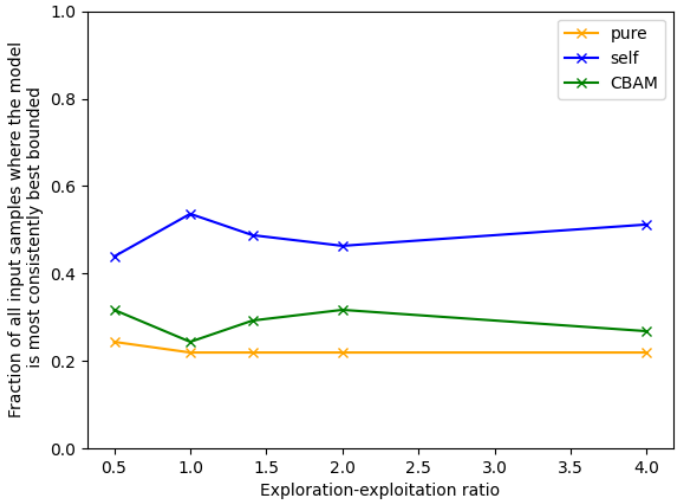

Figure 5.4: Graph showing the change in the *most consistently best bounded* model as we varied R. For each R value we tested, we plot the fraction of samples for which a model was most consistently best bounded among all three models.

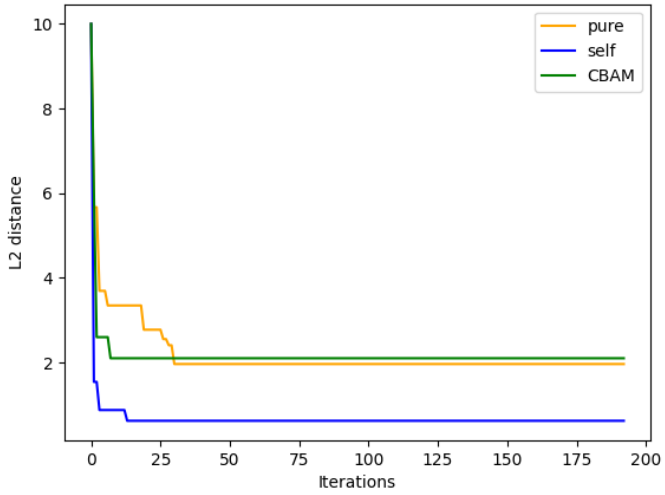

Figure 5.5: Graph of the best upper bound in terms of L_2 distance against the number of iterations of MCTS, for the test input indexed 1279. The parameters are $n_{features} = 10$ and $R = \sqrt{2}$, and the feature extraction method is grey-box.

the fraction at $n_{features} = 10$ is comparable to that at $n_{features} = 2$.

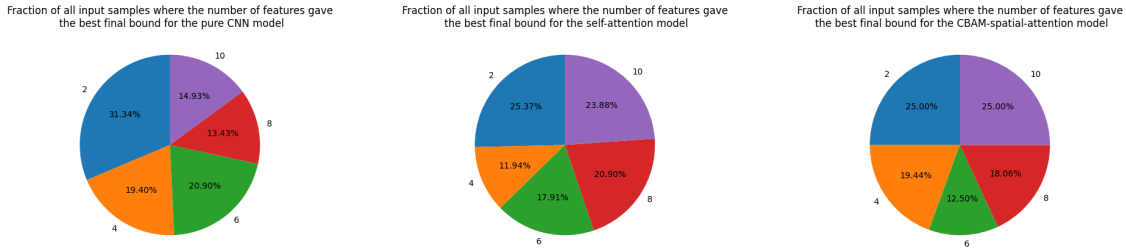

Figure 5.6: Pie charts showing, for a fixed model, the fraction of samples for which each $n_{features}$ value gave the best final bound out of all the $n_{features}$ values we tested. The models are pure (left), self-attention (middle), and CBAM-spatial-attention (right) respectively.

Figure 5.7 shows the relative performance among the different values of R for a given model. For the pure model and the self-attention model, $R = \sqrt{2}$ took the largest fraction, followed closely by $R = 4.0$. For the CBAM-spatial-attention model, $R = \sqrt{2}$ gave the best final bound for the largest fraction.

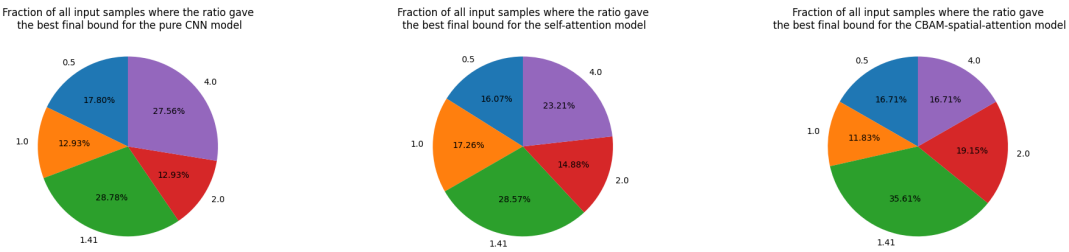

Figure 5.7: Bar charts showing, for a fixed model, the fraction of samples for which each R value gave the best final bound out of all the R values we tested. The models are pure (left), self-attention (middle), and CBAM-spatial-attention (right) respectively.

Fixing the model, Figure 5.8 shows that the grey-box method obtained the best final bound for the larger fraction of samples for the pure and the self-attention models, whereas the black-box method had the larger fraction for the CBAM-spatial-attention model. Nevertheless, the two methods' fractions are very close.

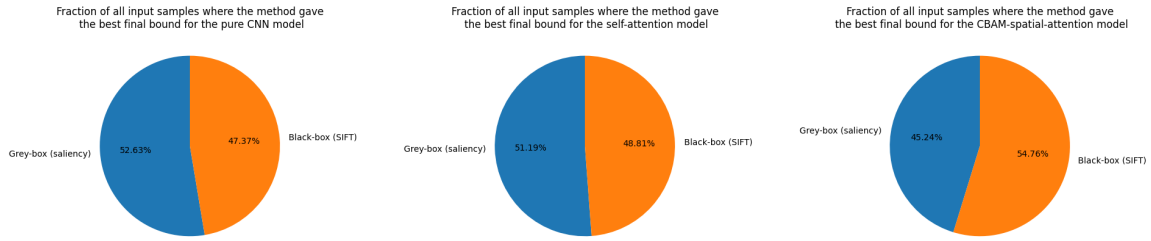

Figure 5.8: Pie charts showing, for a fixed model, the fraction of samples for which each feature extraction method gave the best final bound between the two methods. When the two methods gave the same final bound for an input sample, we split the count evenly as if each method gave the best final bound for half a sample. The models are pure (left), self-attention (middle), and CBAM-spatial-attention (right) respectively.

5.1.2 Lower bounds

Since the A* search is deterministic and independent of the variables we identified, we only performed one set of experiments running the A* search for each model and each input sample. We chose $\tau = 0.01$ as the manipulation magnitude, which does not provide the theoretical guarantees in Chapter 2 since we do not know the Lipschitz constants of the models. Hence, the output of the algorithm may not be real lower bounds, but we can still look at the relative performance of the algorithm on the different models.

Similar to what we observed about the upper bounds, the performance depended much on the inputs. The pure model obtained the best final bound for 39.15% of the samples, the self-attention model 37.98%, and the CBAM-spatial-attention model 22.87%. We observe similar fractions for the most consistently best bounded models. The pure model was the most consistently best bounded for 45.74% of the samples, the self-attention model 36.43%, and the CBAM-spatial-attention model 17.83%.

We focus on a specific input, shown in Figure 5.9, to observe the convergence trends of its bounds. Figure 5.10 shows the convergence trends of the upper bounds and the lower bounds for the input, with some sample images generated in the searches shown in Figure 5.11.

Figure 5.9: The test input indexed 7268.

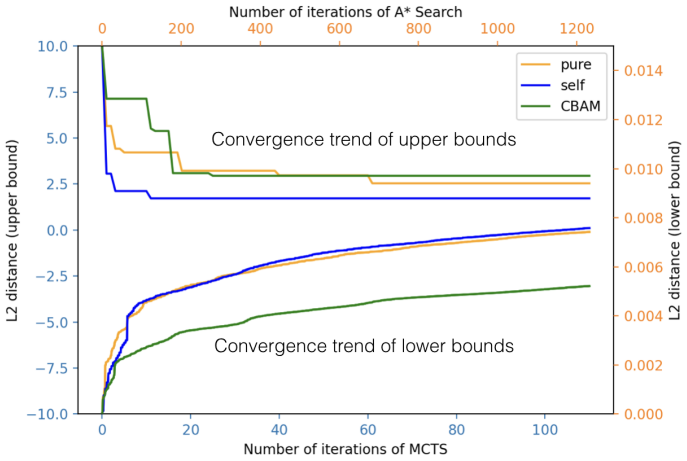

Figure 5.10: The convergence trends of the bounds for the test input indexed 7268. Note the different scales used for the upper bounds and the lower bounds.

We can see both the upper and the lower bounds converging as the experiment runs for more iterations, but the improvement slows down over time. Moreover, the best lower bound found by the A* search is still orders of magnitude smaller than the best upper bound found by MCTS. We can see from Figure 5.11 that the progress made in the A* search is hardly visible to humans. The slow progress in the A* search is partly attributed to the small value of τ, so if τ has to be small to fulfil the theoretical requirement, the current algorithms cannot provide tight bounds in a reasonable amount of time.

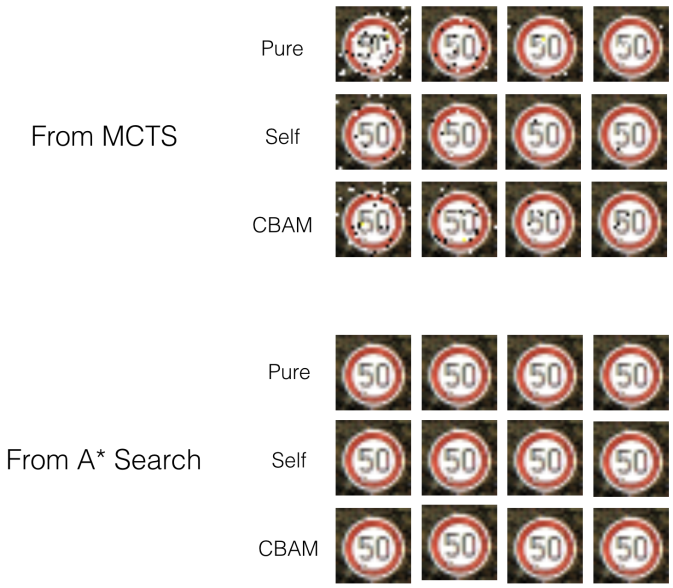

Figure 5.11: Images generated in the searches for the bounds for the test input indexed 7268, showing the progress in each search.

5.2 Features and manipulations

The saliency-guided grey-box feature extraction depends on the models, and the pixels are ranked before partitioning such that the first feature contains pixels that the output is the most sensitive to. The method identified the most important features in the foreground for most inputs, as shown in Figure 5.12. When the adversarial examples are close to the original input, the manipulations that gave the closest adversarial examples found in the experiments tend to occur in the most important features. However, for input samples whose closest adversarial examples are further away, the manipulations occur in many different features. We can interpret the manipulations as the differences between the adversarial example and the original input, so we need more manipulations to reach an adversarial example that is further away from the original input.

The black-box feature extraction is independent of the models, and the features are not ranked by any order. For many input samples, the manipulations occur in different features

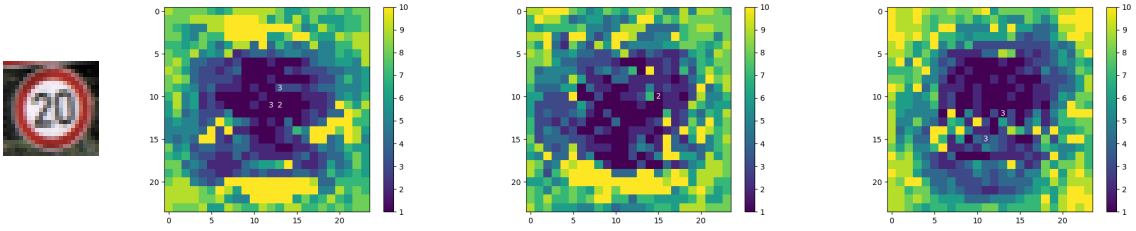

Figure 5.12: Diagrams showing the features of the input indexed 1279 (shown on the left), resulting from the grey-box feature extraction with $n_{features} = 10$. The feature diagrams are for the pure (left), the self-attention (right), and the CBAM-spatial-attention (right) models respectively. The first feature is the one with the darkest colour. The white number on a pixel indicates the number of channels in the pixel that are manipulated to obtain the closest adversarial example (found by MCTS using the default parameters).

like those shown in Figure 5.13, and sometimes they are almost evenly distributed across the features.

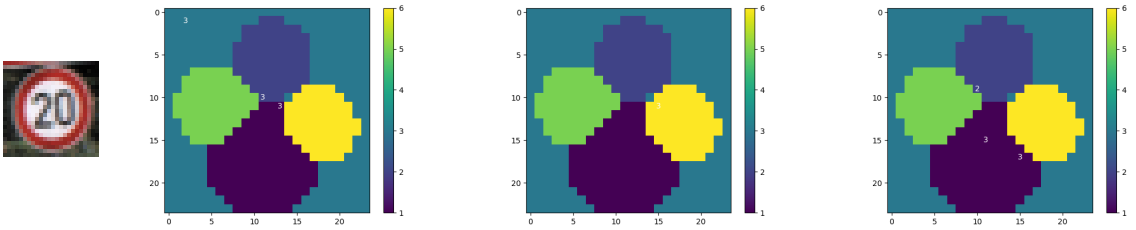

Figure 5.13: Diagrams showing the features of the input indexed 1279, resulting from the black-box feature extraction. The feature diagrams are for the pure (left), the self-attention (right), and the CBAM-spatial-attention (right) models respectively.

We also found that for some inputs, the pixels in the most important feature from the self-attention model are more scattered, as shown in Figure 5.14. This could suggest that the self-attention model identified multiple things in the input to focus on, but not necessarily the important ones for the classification task. This is corroborated by the finding in [LCKK22], which suggests that attention models may not be more robust than pure models.

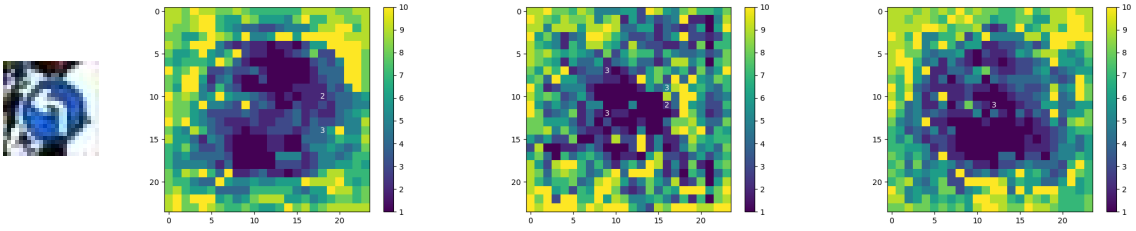

Figure 5.14: Diagrams showing the features of the input indexed 8320, resulting from the grey-box feature extraction with $n_{features} = 10$. The features are for the pure (left), the self-attention (right), and the CBAM-spatial-attention (right) models respectively.

5.3 Adversarial examples

MCTS found adversarial examples for all models for 41 out of the 43 input samples. The numbers of adversarial examples found throughout an experiment range from fewer than 10 to a few hundred, but in most experiments more than a hundred adversarial examples are found. In all sets of experiments, the self-attention model is the model that obtained the most adversarial examples for the majority of the input samples.

Figure 5.15 shows the best adversarial examples found in the experiments for three input samples. The adversarial examples are very close to the original input, but the models classified them into the wrong categories. In particular, for the input indexed 1279 whose true class is "speed limit 20", the pure model classified the adversarial example into "speed limit 30" with confidence 0.643, the self-attention model classified the adversarial example into "speed limit 30" with confidence 0.734, and the CBAM-spatial-attention model classified the adversarial example into "speed limit 80" with confidence 0.956. We observe that the adversarial examples against the attention models are sometimes more dangerous than those against the pure model. For instance, we can see in Figure 5.15 that the adversarial examples for the image indexed 1279 against the self-attention model and for the image indexed 8320 against the CBAM-spatial-attention model look almost identical to their original images.

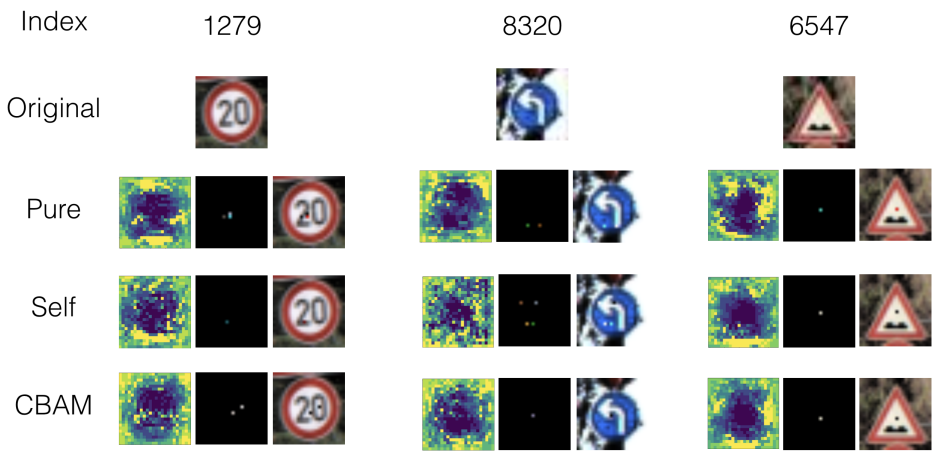

Figure 5.15: The closest adversarial examples of three inputs found by MCTS for the three models. Each group of three pictures show, from left to right, the features extracted (where pixels with the same colour are in the same partition), the difference between the adversarial example and the original image, and the adversarial example.

While we varied the values of $n_{features}$ and R, we observe that the self-attention model always gave the most adversarial examples among the three models for most input samples, as shown in Figure 5.16 and Figure 5.17. The same is true when we changed the feature extraction method to the black-box method. With our previous observation that the self-attention model gives the least final upper bounds for the largest fraction of inputs, this could indicate that for most inputs, the self-attention model has more adversarial examples that are close to the original image compared to the other models.

5.4 Running time

An iteration of MCTS can take only a few seconds if the simulations reach terminal nodes quickly by finding adversarial examples; but it may take up to a few minutes if the simulations only terminate due to reaching a point outside $\eta(\alpha, L_2, 10)$. The total time taken increases with the number of simulations we require for each node, and the relationship is expected to be proportional since the simulations are independent.

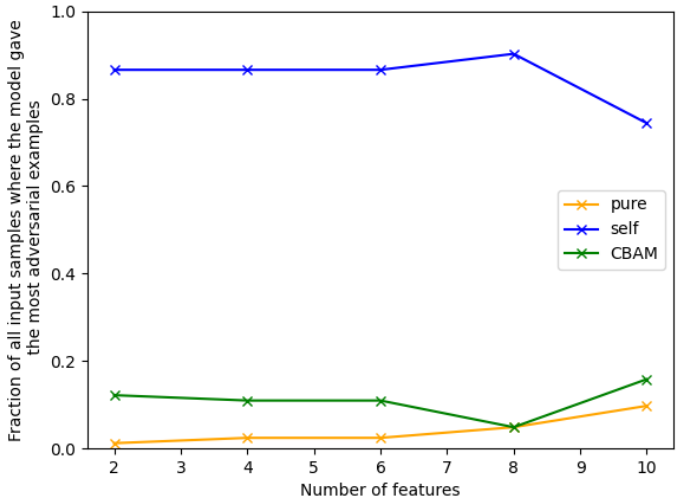

Figure 5.16: Graph showing the change in the model with the most adversarial examples found as we varied $n_{features}$. For each $n_{features}$ value we tested, we plot the fraction of samples for which a model gave the most adversarial examples among all three models.

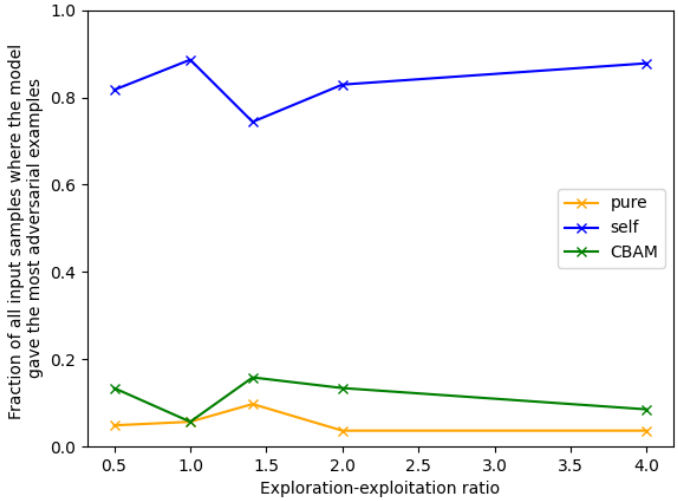

Figure 5.17: Graph showing the change in the model with the most adversarial examples found as we varied R. For each R value we tested, we plot the fraction of samples for which a model gave the most adversarial examples among all three models.

At the beginning of the A* search, an iteration takes seconds, but the time taken by an iteration increases to hours deeper down the search tree, as the number of nodes to be explored increases exponentially with the depth of the tree, and we need to select one with the best value under the evaluation function for the next expansion. The asymptotic running time is consistent with that described in [Wu20].

Chapter 6

Conclusion

In this project, we improved the `DeepGame` tool from [WWR+18], and we constructed and trained two attention models with architectures from [BZV+19, WPLK18] using the GTSRB dataset. We then demonstrated the use of `DeepGame` on the models for bounding pointwise robustness and finding adversarial examples, and compared its performances on the attention models with the performance on the pure CNN model. We identified a few parameters that could affect the upper bounds found by the MCTS algorithm, namely the number of features extracted, the exploration-exploitation ratio, and the feature extraction method, and we experimented on different values of the parameters and compared the results.

We found that for the inputs we sampled, the relative performance of `DeepGame` on the different models, for both the upper and the lower bounds, depends largely on the inputs. Nevertheless, for a given model, specific parameter values could give better upper bounds for a large fraction of the input samples. We also showed that the MCTS algorithm can find many adversarial examples for the attention models, comparable to its performance on the pure model, and the adversarial examples can be useful for guiding the training of neural networks for greater robustness. By observing the features extracted and manipulated, we found evidence that the saliency-guided grey-box feature extraction method could extract

meaningful features, and that could help guide the MCTS search. Meanwhile, we observed that the self-attention model obtained the least final upper bounds and the most adversarial examples in MCTS for the largest fraction of input samples, while `DeepGame`'s performance on the CBAM-spatial-attention model is closer to that on the pure CNN model. This suggests that the self-attention model may be less robust for the image classification task, whereas CBAM-spatial-attention which has a structure closer to CNNs may be more suitable for such tasks.

Chapter 7

Reflection and future work

Given the time constraint of the project, we could only try two different attention models on 43 samples. The sample size is too small for us to make any statistical observation, and we can only discuss observations specific to the models and the input samples we used. Hence, future research may investigate other types of attention models and perform experiments on more input samples and on different datasets to reach a more general conclusion.

Moreover, the problem of finding the finite maximum safe radius (FMSR) is intractable, so we need to use heuristics in the search problem for greater efficiency. We used existing heuristics in both the MCTS algorithm and the A* search algorithm to guide our search for bounds on the FMSR, but there may be better heuristics for the task. The heuristic for the A* search needs a Lipschitz constant of the model so that the search is guaranteed to produce lower bounds, but there is currently no existing method to find Lipschitz constants of the attention models, so we could not have the theoretical guarantee. Finding better heuristics and building models with provably small Lipschitz constants can both contribute to the efficiency of the tool.

Another issue we did not resolve is finding a standard for a fair comparison. The parallelization improved the efficiency, but also introduced more uncertainty into the running time

allocated by the operating system to each run of the experiment, so it may not be fair to compare the results given the same time constraint; the time taken by each iteration depends much on the model and the input, so comparison based on the same number of iterations may not be fair either. There may be other standards of comparison whose fairness could be more justifiable.

On a larger scale, a limitation of this project is that it focuses on pointwise robustness defined by a specific search problem, and this notion of robustness can be very dependent on the input. The performance of attention models under other notions of robustness, such as global robustness, is another area to explore.

Chapter 8

References

[BZV+19] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3286–3295, 2019.

[CMN19] Ştefan Cobzaş, Radu Miculescu, and Adriana Nicolae. *Lipschitz functions [electronic resource]*. Lecture notes in mathematics (Springer-Verlag) ; 2241. Cham, Switzerland : Springer, Cham, Switzerland, 2019.

[FRH+19] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning [electronic resource]*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, Cambridge, Massachusetts, 2016.

[GLB+18] Mohamad Gharib, Paolo Lollini, Marco Botta, Elvio Amparore, Susanna Donatelli, and Andrea Bondavalli. On the safety of automotive systems incorporating machine learning based components: A position paper. 06 2018.

[HKR+20] 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. *Computer Science Review*, 37:100270, 2020.

[HKWT18] Andreas Holzinger, Peter Kieseberg, Edgar Weippl, and A. Min Tjoa. Current advances, trends and challenges of machine learning and knowledge extraction: From machine learning to explainable ai. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl, editors, *Machine Learning and Knowledge Extraction*, pages 1–8, Cham, 2018. Springer International Publishing.

[JSZ+15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.

[KSZQ20] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *The Artificial intelligence review*, 53(8):5455–5516, 2020.

[LCKK22] Hsuan-Cheng Liao, Chih-Hong Cheng, Maximilian Kneissl, and Alois Knoll. Robustness verification for attention networks using mixed integer programming. *arXiv preprint arXiv:2202.03932*, 2022.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[RHK18] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242*, 2018.

[SSSI11] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: A multi-class classification competition. In *The 2011 International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[WPLK18] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[Wu20] Min Wu. *Robustness Evaluation of Deep Neural Networks with Provable Guarantees*. PhD thesis, University of Oxford, 2020.

[WWR+18] Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. A game-based approximate verification of deep neural networks with provable guarantees. *CoRR*, abs/1807.03571, 2018.

Chapter 9

Appendix

9.1 Model architectures and training parameters

9.1.1 Pure CNN model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 22, 22, 64)        1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 20, 64)        36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 128)         147584
_________________________________________________________________
max_pooling2d_1              (None, 3, 3, 128)         0
(MaxPooling2D)
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 256)               295168
_________________________________________________________________
```

```
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_2 (Dense)              (None, 43)                11051
_________________________________________________________________
softmax (Softmax)            (None, 43)                0
=================================================================
Total params: 632,171
Trainable params: 632,171
Non-trainable params: 0
_________________________________________________________________
```

- Batch size = 128
- Number of epochs = 50
- Optimizer = Root Mean Squared propagation (`RMSprop`) (learning rate = 0.0001, decay=10^{-6})
- Loss function = `categorical_crossentropy`
- Training accuracy = 95.58%
- Testing accuracy = 93.27%

9.1.2 Self-attention model

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 24, 24, 3)]  0
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 24, 24, 48)   192         input_1[0][0]
__________________________________________________________________________________________________
tf.split                        [(None, 24, 24, 16), 0           conv2d_1[0][0]
                                 (None, 24, 24, 16),
                                 (None, 24, 24, 16)]
__________________________________________________________________________________________________
tf.reshape_1                    (None, 576, 16)      0           tf.split[0][1]
```

```
tf.reshape                      (None, 576, 16)      0           tf.split[0][0]
__________________________________________________________________________________________________
tf.linalg.matmul                (None, 576, 576)     0           tf.reshape_1[0][0]
                                                                 tf.reshape[0][0]
__________________________________________________________________________________________________
tf.math.multiply                (None, 576, 576)     0           tf.linalg.matmul[0][0]
__________________________________________________________________________________________________
tf.nn.softmax                   (None, 576, 576)     0           tf.math.multiply[0][0]
__________________________________________________________________________________________________
tf.reshape_2                    (None, 576, 16)      0           tf.split[0][2]
__________________________________________________________________________________________________
tf.linalg.matmul_1              (None, 576, 16)      0           tf.nn.softmax[0][0]
                                                                 tf.reshape_2[0][0]
__________________________________________________________________________________________________
tf.reshape_3                    (None, 24, 24, 16)   0           tf.linalg.matmul_1[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 24, 24, 48)   1344        input_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 24, 24, 16)   272         tf.reshape_3[0][0]
__________________________________________________________________________________________________
tf.concat                       (None, 24, 24, 64)   0           conv2d[0][0]
                                                                 conv2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization             (None, 24, 24, 64)   256         tf.concat[0][0]
(BatchNormalization)
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 22, 22, 64)   36928       batch_normalization[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 11, 11, 64)   0           conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 11, 11, 96)   6240        max_pooling2d[0][0]
__________________________________________________________________________________________________
tf.split_1                      [(None, 11, 11, 32), 0           conv2d_5[0][0]
                                 (None, 11, 11, 32),
                                 (None, 11, 11, 32)]
__________________________________________________________________________________________________
tf.reshape_5                    (None, 121, 32)      0           tf.split_1[0][1]
__________________________________________________________________________________________________
```

```
tf.reshape_4                    (None, 121, 32)     0           tf.split_1[0][0]
__________________________________________________________________________________________________
tf.linalg.matmul_2              (None, 121, 121)    0           tf.reshape_5[0][0]
                                                                tf.reshape_4[0][0]
__________________________________________________________________________________________________
tf.math.multiply_1              (None, 121, 121)    0           tf.linalg.matmul_2[0][0]
__________________________________________________________________________________________________
tf.nn.softmax_1                 (None, 121, 121)    0           tf.math.multiply_1[0][0]
__________________________________________________________________________________________________
tf.reshape_6                    (None, 121, 32)     0           tf.split_1[0][2]
__________________________________________________________________________________________________
tf.linalg.matmul_3              (None, 121, 32)     0           tf.nn.softmax_1[0][0]
                                                                tf.reshape_6[0][0]
__________________________________________________________________________________________________
tf.reshape_7                    (None, 11, 11, 32)  0           tf.linalg.matmul_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 11, 11, 96)  55392       max_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 11, 11, 32)  1056        tf.reshape_7[0][0]
__________________________________________________________________________________________________
tf.concat_1                     (None, 11, 11, 128) 0           conv2d_4[0][0]
                                                                conv2d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_1           (None, 11, 11, 128) 512         tf.concat_1[0][0]
(BatchNormalization)
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 9, 9, 128)   147584      batch_normalization_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 4, 4, 128)   0           conv2d_7[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 2048)        0           max_pooling2d_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)         524544      flatten[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 256)         0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)         65792       dropout[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 43)          11051       dense_1[0][0]
```

```
____________________________________________________________________________________________
softmax (Softmax)               (None, 43)           0           dense_2[0][0]
============================================================================================
Total params: 851,163
Trainable params: 850,779
Non-trainable params: 384
____________________________________________________________________________________________
```

- Batch size = 128
- Number of epochs = 50
- Optimizer = `RMSprop` (learning rate = 0.0001, decay=10^{-6})
- Loss function = `categorical_crossentropy`
- Training accuracy = 98.73%
- Testing accuracy = 94.73%

9.1.3 CBAM-spatial-attention model

```
____________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
============================================================================================
input_1 (InputLayer)            [(None, 24, 24, 3)]  0
____________________________________________________________________________________________
conv2d (Conv2D)                 (None, 22, 22, 64)   1792        input_1[0][0]
____________________________________________________________________________________________
tf.math.reduce_mean             (None, 22, 22)       0           conv2d[0][0]
____________________________________________________________________________________________
tf.math.reduce_max              (None, 22, 22)       0           conv2d[0][0]
____________________________________________________________________________________________
tf.reshape                      (None, 22, 22, 1)    0           tf.math.reduce_mean[0][0]
____________________________________________________________________________________________
tf.reshape_1                    (None, 22, 22, 1)    0           tf.math.reduce_max[0][0]
____________________________________________________________________________________________
tf.concat                       (None, 22, 22, 2)    0           tf.reshape[0][0]
                                                                 tf.reshape_1[0][0]
```

```
conv2d_1 (Conv2D)               (None, 22, 22, 1)    99          tf.concat[0][0]
__________________________________________________________________________________________________
tf.tile                         (None, 22, 22, 64)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
tf.math.multiply                (None, 22, 22, 64)   0           conv2d[0][0]
                                                                 tf.tile[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 22, 22, 64)   0           conv2d[0][0]
                                                                 tf.math.multiply[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 20, 20, 64)   36928       add[0][0]
__________________________________________________________________________________________________
tf.math.reduce_mean_1           (None, 20, 20)       0           conv2d_2[0][0]
__________________________________________________________________________________________________
tf.math.reduce_max_1            (None, 20, 20)       0           conv2d_2[0][0]
__________________________________________________________________________________________________
tf.reshape_2                    (None, 20, 20, 1)    0           tf.math.reduce_mean_1[0][0]
__________________________________________________________________________________________________
tf.reshape_3                    (None, 20, 20, 1)    0           tf.math.reduce_max_1[0][0]
__________________________________________________________________________________________________
tf.concat_1                     (None, 20, 20, 2)    0           tf.reshape_2[0][0]
                                                                 tf.reshape_3[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 20, 20, 1)    99          tf.concat_1[0][0]
__________________________________________________________________________________________________
tf.tile_1                       (None, 20, 20, 64)   0           conv2d_3[0][0]
__________________________________________________________________________________________________
tf.math.multiply_1              (None, 20, 20, 64)   0           conv2d_2[0][0]
                                                                 tf.tile_1[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 20, 20, 64)   0           conv2d_2[0][0]
                                                                 tf.math.multiply_1[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 10, 10, 64)   0           add_1[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 8, 8, 128)    73856       max_pooling2d[0][0]
__________________________________________________________________________________________________
tf.math.reduce_mean_2           (None, 8, 8)         0           conv2d_4[0][0]
```

```
tf.math.reduce_max_2            (None, 8, 8)         0           conv2d_4[0][0]
__________________________________________________________________________________________________
tf.reshape_4                    (None, 8, 8, 1)      0           tf.math.reduce_mean_2[0][0]
__________________________________________________________________________________________________
tf.reshape_5                    (None, 8, 8, 1)      0           tf.math.reduce_max_2[0][0]
__________________________________________________________________________________________________
tf.concat_2                     (None, 8, 8, 2)      0           tf.reshape_4[0][0]
                                                                 tf.reshape_5[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 8, 8, 1)      99          tf.concat_2[0][0]
__________________________________________________________________________________________________
tf.tile_2                       (None, 8, 8, 128)    0           conv2d_5[0][0]
__________________________________________________________________________________________________
tf.math.multiply_2              (None, 8, 8, 128)    0           conv2d_4[0][0]
                                                                 tf.tile_2[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 8, 8, 128)    0           conv2d_4[0][0]
                                                                 tf.math.multiply_2[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 6, 6, 128)    147584      add_2[0][0]
__________________________________________________________________________________________________
tf.math.reduce_mean_3           (None, 6, 6)         0           conv2d_6[0][0]
__________________________________________________________________________________________________
tf.math.reduce_max_3            (None, 6, 6)         0           conv2d_6[0][0]
__________________________________________________________________________________________________
tf.reshape_6                    (None, 6, 6, 1)      0           tf.math.reduce_mean_3[0][0]
__________________________________________________________________________________________________
tf.reshape_7                    (None, 6, 6, 1)      0           tf.math.reduce_max_3[0][0]
__________________________________________________________________________________________________
tf.concat_3                     (None, 6, 6, 2)      0           tf.reshape_6[0][0]
                                                                 tf.reshape_7[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 6, 6, 1)      99          tf.concat_3[0][0]
__________________________________________________________________________________________________
tf.tile_3                       (None, 6, 6, 128)    0           conv2d_7[0][0]
__________________________________________________________________________________________________
tf.math.multiply_3              (None, 6, 6, 128)    0           conv2d_6[0][0]
                                                                 tf.tile_3[0][0]
```

```
__________________________________________________________________________________________________
add_3 (Add)                     (None, 6, 6, 128)    0           conv2d_6[0][0]
                                                                 tf.math.multiply_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 3, 3, 128)    0           add_3[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1152)         0           max_pooling2d_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          295168      flatten[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 256)          0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          65792       dropout[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 43)           11051       dense_1[0][0]
__________________________________________________________________________________________________
softmax (Softmax)               (None, 43)           0           dense_2[0][0]
==================================================================================================
Total params: 632,567
Trainable params: 632,567
Non-trainable params: 0
__________________________________________________________________________________________________
```

- Batch size = 128
- Number of epochs = 50
- Optimizer = `RMSprop` (learning rate = 0.0001, decay=10^{-6})
- Loss function = `categorical_crossentropy`
- Training accuracy = 98.00%
- Testing accuracy = 94.66%

9.2 `DeepGame` parameters

9.2.1 MCTS

- Number of simulations per iteration = 3
- Given the input sample α, search in $\eta(\alpha, L_2, 10)$
- $\tau = 1$

9.2.2 A* search

- Given the input sample α, search in $\eta(\alpha, L_2, 10)$
- $\tau = 0.01$

9.3 Hardware and software platforms

9.3.1 Hardware

- 6 NVIDIA 2080Ti GPUs
- 4 20-core Intel Core Xeon 6230's

9.3.2 Software

- Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-147-generic x86_64)
- Conda 4.10.3
- Python 3.9.7 with tensorflow-gpu 2.4.1