Formal Verification of Tree Ensembles against Real-World Composite Geometric Perturbations

Valency Oscar Colaco[1,*], Simin Nadjm-Tehrani[1]

[1]Department of Computer and Information Science, Linköping University, Sweden

Abstract

Since machine learning components are now being considered for integration in safety-critical systems, safety stakeholders should be able to provide convincing arguments that the systems are safe for use in realistic deployment settings. In the case of vision-based systems, the use of tree ensembles calls for formal stability verification against a host of composite geometric perturbations that the system may encounter. Such perturbations are a combination of an affine transformation like rotation, scaling, or translation and a pixel-wise transformation like changes in lighting. However, existing verification approaches mostly target small norm-based perturbations, and do not account for composite geometric perturbations. In this work, we present a novel method to precisely define the desired stability regions for these types of perturbations. We propose a feature space modelling process that generates abstract intervals which can be passed to VoTE, an efficient formal verification engine that is specialised for tree ensembles. Our method is implemented as an extension to VoTE by defining a new property checker. The applicability of the method is demonstrated by verifying classifier stability and computing metrics associated with stability and correctness, i.e., robustness, fragility, vulnerability, and breakage, in two case studies. In both case studies, targeted data augmentation pre-processing steps were applied for robust model training. Our results show that even models trained with augmented data are unable to handle these types of perturbations, thereby emphasising the need for certified robust training for tree ensembles.

Keywords

Machine Learning, Formal Verification, Tree Ensembles, Composite Perturbations, Geometric Perturbations, Random Forests, Gradient Boosting Machines, Semantic Perturbations, Stability, Robustness, Trustworthy AI, Trustworthy Computing

1. Introduction

Recent advancements in machine learning are now being considered for integration in safety-critical systems like medical equipment, critical infrastructure, and transport networks (autonomous vehicles, avionics, spaceflights) where software flaws could be detrimental to humans and the environment. While these systems are subject to strict regulations, a lack of evidence that the machine learning (ML) models deployed in these systems are safe for operation, can directly affect their trustworthiness. Considering the inherently probabilistic nature of these models, standard industry practices like software testing are often unsuitable to provide any guarantees about operational safety. In this context, to help ensure trustworthiness, Artificial Intelligence (AI) systems can benefit from scrutiny through formal methods. Formal verification can be a tool to analyse the safety properties of software operating over a multi-dimensional space. Evidence based on formal verification can be used in a safety assurance case, which is a structured record of reasoning with compelling and comprehensible arguments for a system being acceptably safe in a given context [25].

The formal methods community has recently been exploring the robustness properties of vision-based AI systems used in autonomous vehicles [5]. An important role of these vision-based systems is sending correct information about road signs to the vehicle's control system. However, the road signs encountered in the real world may appear different to that in the test set. Real-world phenomena like the banking of roads cause changes to the road surface topography in which the outer edges are raised at an angle above the inner edge to provide the necessary centripetal force to the vehicles for safe turns. This phenomenon, combined with rapid changes in lighting (time of day, light reflections from vehicles), and the fact that road signs have reflective elements, can produce different versions of the road signs. Along with being rotated, parts of these signs may appear lighter and darker depending on the light interactions (absorption, reflection, and so on). The same principle can be applied to other geometric transformations like scaling, in which the road sign appears zoomed out when the vehicle is at a distance, and translation, in which the road sign gradually moves out of the camera frame as it is approached by the vehicle. While the different versions of these road signs are correctly interpretable by humans, an ML model could misread these signs and the effects

The AAAI-23 Workshop on Artificial Intelligence Safety (SafeAI 2023), Feb 13-14, 2023, Washington, D.C., US

*Corresponding author.

valency.colaco@liu.se (V. O. Colaco);

simin.nadjm-tehrani@liu.se (S. Nadjm-Tehrani)

0000-0001-6405-4794 (V. O. Colaco); 0000-0002-1485-0802 (S. Nadjm-Tehrani)

could be disastrous. This calls for formal verification of these models to ensure classifier stability in the presence of composite geometric perturbations that the system may encounter in real-world deployment settings.

In this paper, we present a method to precisely capture these perturbations as abstract intervals to verify tree-ensemble based classifiers against specified safety properties. Since our goal is to support safety arguments in assurance cases, the verifiability of the models is of paramount importance. The simple structure of the tree-based models makes them easier to systematically analyze in the formal verification process. However, larger models may still prove hard to verify due to the combinatorial explosion. We present a method where composite geometric perturbations are applied to input data and shown to prove a lack of stability even in the presence of data augmentation during training. This is demonstrated by applying our method to digit/character recognition problems. The contributions of this paper are as follows,

- A method to capture composite geometric perturbations based on real-world phenomena (like the combined effects of axial rotations, scaling, or translation together with changes in lighting) in the interval domain through feature space modelling for vision-based systems.
- Realisation of the method, implemented as an extension to the Verifier of Tree Ensembles [3] (VoTE), a formal verification tool that uses abstract interpretation, along with the application of the method to two case studies commonly found in the literature.

The rest of the paper is structured as follows. Section 2 discusses related works on the verification of tree ensembles against different classes of perturbations. Section 3 presents the preliminaries on tree-ensembles, the VoTE toolsuite, and composite geometric perturbations. Section 4 presents our abstraction function and feature space modelling process along with the implementation in VoTE. Section 5 presents the application of the method to two case studies commonly found in the literature. Section 6 concludes the paper and summarizes the learnings.

2. Related Works

Recent machine learning advances are gradually finding their way into safety-critical systems. However, there is a lack of verification techniques to build formal arguments for operational safety in such systems. There has been extensive research on formal verification of l_p-norm bounded perturbations (mathematical distances that define the perturbations) in neural networks. A recent survey [11] gives a broad overview of the safety and trustworthiness of deep neural networks with a specific focus on topics like verification, testing, adversarial defence, and interpretability.

Engstrom et al. [12], show that state of the art neural network classifiers are vulnerable to natural classes of perturbations, especially simple translations and rotations for a significant fraction of their inputs. Another interesting observation they made is that data augmentation does not necessarily provide a significant boost to robustness, which we also confirm for tree ensembles in this paper.

Pei et al. [13] and Mohapatra et al. [14] propose verification frameworks called VERIVIS and Semantify-NN for evaluating the robustness of machine learning systems. The VERIVIS framework checks for violations of the parameter space (like the range of angles) when the inputs are subjected to semantic perturbations while the Semantify-NN framework converts semantic perturbations to l_p-norm based threat models thereby enabling the use of any l_p-norm based verification method for robustness verification of deep neural networks. Balunović et al. [15] study the certification of neural networks against natural geometric transformations like rotation and scaling. They use sampling, interpolation and Lipschitz optimisation to find the bounds for their certification process. The authors implement their certification framework in a system called DEEPG that uses a restricted Polyhedra with custom approximations for the operations used in several geometric transformations. Our approach, however, uses a feature space modelling process in conjunction with an abstraction function to get precise upper and lower bounds for the inputs as they are perturbed by real-world elements.

Wu et al. [16], study the certification of deep neural networks against real-world distribution shifts. They train a generative model to learn perturbations from the data to improve robustness precision. Yang et al. [17] mention that geometric image transformations that arise in the real world, such as scaling and rotation, have been shown to easily deceive deep neural networks. They propose a training formulation known as Certified Geometric Training (CGT) that improves the deterministic certified robustness of neural networks with respect to geometric transformations. Gowal et al. [19], use generative models to systematically transfer image attributes that are likely to be misclassified across image instances to achieve better robustness. They also mention that l_p-norm bounded perturbations do not necessarily cover plausible real-world variations that preserve the semantics of the input. We agree

that composite geometric perturbations based on real-world phenomena (that preserve the semantics of the input) are outside the l_p-norm reach. Hence, we deploy a feature space modelling process for defining the multidimensional perturbation regions. Also, while using generative models can help with robust training, this in itself cannot be used for building up a safety case in systems that use ML components. Our approach aims to provide formal proof-based evidence to support safety assurances in these systems.

Outside formal verification and robust training, significant research has been done to tackle the issue of real-world perturbations like axial rotations in images in the form of rotation-invariant networks [20] and rotation-equivariant networks [21]. In Scher et al. [10], the authors formally define real-world perturbations and differentiate them between adversarial perturbations and distribution shifts. They model the perturbations around a point as a probability distribution and succeed in the estimation of robustness using Monte Carlo Sampling for low to medium dimensional data. While their method provides probabilistic guarantees, our approach aims to provide formal deterministic guarantees which can form the basis for reasoning about safety in safety-critical systems.

3. Preliminaries

In this section, we present the background knowledge on tree-based ensembles, semantic perturbations, interpolated/geometric and pixel-wise transformations, composite geometric perturbations based on real-world phenomena, and the toolsuite VoTE [3].

3.1. Decision Trees

Decision trees are employed as prediction models in machine learning and can be used in vision-based systems to solve image classification problems (among other applications). A decision tree model contains rules to predict the output class and implements a prediction function $t : X^n \rightarrow \mathbb{R}^m$ that maps disjoint sets of points $X_i \subset X^n$ to a single output point $\overline{y_i} \in \mathbb{R}^m$, i.e.,

$$t(\overline{x}) = \begin{cases} (y_{1,1}, ..., y_{1,m}) & \overline{x} \in X_1 \\ \vdots & \\ (y_{k,1}, ..., y_{k,m}) & \overline{x} \in X_k \end{cases} \tag{1}$$

Where k is the number of disjoint sets and $X^n = \bigcup_{i=1}^{k} X_i$

The n-dimensional input domain X^n includes elements $\overline{x}$ as tuples in which each input variable x_i captures some feature of interest of the system [3]. The tree structure is evaluated in a top-down manner, where decision functions determine which path to take towards the leaves. Each internal node in the decision tree is associated with a decision function that recursively splits the input space, thereby separating regions from each other. When a leaf is hit, the output $\overline{y} \in \mathbb{R}^m$ associated with the leaf becomes the output associated with the input from which the top-down evaluation started.

In this paper, we only consider binary trees with linear decision functions with one input variable, which Irsoy et al. [4] call univariate hard decision trees. State-of-the-art implementations of tree-based ensembles typically use univariate hard decision trees, for example, scikit-learn [6] and CatBoost [7].

3.2. Random Forests

Decision trees are known to suffer from a phenomenon called overfitting in which the models are fitted so tightly to their training data that they memorize the data itself instead of learning the patterns associated with the data in order to make generalized predictions. To counteract these issues with decision trees concerning bias and variance, Breiman [1] proposes Random Forests.

A random forest $f : X^n \rightarrow \mathbb{R}^m$ is an ensemble of B decision trees that produces outputs by averaging the values emitted by each individual tree. i.e.,

$$f(\overline{x}) = \frac{1}{B} \sum_{b=1}^{B} t_b(\overline{x}) \tag{2}$$

Where t_b is the b^{th} tree in the ensemble

To reduce the correlation between trees, each tree is trained on overlapping random subsets of the training data (with replacement) using techniques like bagging or boosting.

3.3. Gradient Boosting Machines

Similar to Random Forests, Freidman [2] proposes gradient boosting machines that employ several decision trees to create a prediction function. Unlike Random Forests, these trees are trained in a sequential manner with each succeeding tree correcting the errors made by the predecessor tree. The errors are corrected using gradient descent (hence the name). Although conceptually different from random forests in a learning context, these two models have several features in common when performing predictions.

A gradient boosting machine $f : X^n \rightarrow \mathbb{R}^m$ is an

ensemble of B additive decision trees, i.e.,

$$f(\bar{x}) = \sum_{b=1}^{B} t_b(\bar{x}) \tag{3}$$

Where t_b is the b^{th} tree in the ensemble

3.4. Classifier

Decision trees and tree-ensemble models can be used as classifiers to categorize samples from an input domain into one or more classes and assign each sample a label unique to its class. In this paper, we only consider functions that map each point from an input domain to exactly one class. Let $f(\bar{x}) = (y_1, ..., y_m)$ represent a model trained to predict the probability y_i associated with a class i within disjoint regions in the input domain, where m is the number of classes. A classifier $f_c(\bar{x})$ may then be defined as,

$$f_c(\bar{x}) = \underset{i}{argmax}\ y_i \tag{4}$$

3.5. Verifier of Tree Ensembles

VoTE [3] (Verifier of Tree Ensembles) is a toolsuite for formally verifying that tree ensembles comply with specific properties that the user can define through implementing a property checker.

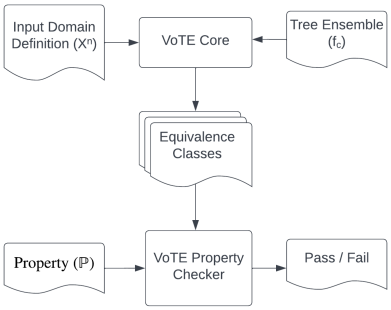

Figure 1: Verifier of Tree Ensembles [3] (VoTE)

The tool is based on abstract interpretation which yields equivalence classes in a tree ensemble, i.e. sets of points in the input domain that yield the same output tuple through an iterative refinement in the interval domain, such that the lowest level of abstract (the concrete) domain is equivalent to an exhaustive search down to all leaves in the model. This technique can be used to verify any property of interest defined within VoTE's property checker. For instance, in [26], VoTE was used to verify versatile application-specific properties of the aircraft collision avoidance system called ACAS Xu [18]. However, in this work, the property checker defined earlier (see Figure 1) checks for stability. The VoTE Core is instantiated from the ensemble subject to verification, $f : X^n \rightarrow R^m$. It takes as input a hyperrectangle defining X^n, and the outcome of the formal verification is *pass* (when the tree ensemble satisfies the concerned property) and *fail* when it does not.

3.6. Semantic Perturbations

As opposed to perturbations that change the value of each feature by a constant value, semantic perturbations take an input image and p parameters to perform a parameterized operation that perturbs the image while preserving its semantic information. Semantics-preserving perturbations can include object-level shifts (like changing the shape or size of an object in a color classification problem), geometric transformations (scaling, rotations) and common image corruptions (fog, blurs) [8]. Most of these perturbations cannot be naturally represented using the l_p-norm ($\| x - x' \|_p \leq \epsilon$) that most robustness verification methods are designed for. Using a large value of ϵ to try and capture these types of perturbations is usually not recommended in practice as the image quality degrades significantly [8]. Hence, we propose a feature space modelling process to precisely capture these multi-dimensional perturbations.

3.7. Geometric Perturbations

Geometric Perturbations are a class of semantic perturbations [8] that involve an affine transformation on each pixel's row and column indices, followed by an interpolation operation [17]. In this paper, we consider three types of affine transformations - rotations, scaling and translation.

Let $T_\theta : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ (in 2D) be an invertible affine transformation (like rotation) parameterized by θ (like the angle of rotation). Let $\phi_x(j)$ and $\phi_y(i)$ be functions that transform i, j pixel indices to x, y co-ordinates with respect to the center of the image. However, as these transformed co-ordinates may not align exactly with the integer-valued pixel indices, bilinear interpolation (I) is necessary. Yang et al. [17] define the general form of a geometric perturbation for an image X as,

$$X'_{i,j} = I(T_\theta^{-1}(\phi_x(j)\ ,\ \phi_y(i))) \tag{5}$$

Where X' is the perturbed image. For each geometric perturbation, they present the inverse transform function T_θ^{-1} which is used to instantiate equation (5),

Rotation, parameterized by angle, $\theta \in [0, 2\pi]$

$$T_\theta^{-1}(x, y) = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} xcos\theta + ysin\theta \\ -xsin\theta + ycos\theta \end{bmatrix} \tag{6}$$

Translation, parameterized by horizontal and vertical shifts $h \in \mathbb{R}$, $v \in \mathbb{R}$

$$T_{h,v}^{-1}(x, y) = \begin{bmatrix} x - h \\ y - v \end{bmatrix} \tag{7}$$

Scaling, parameterized by a scaling factor, $\phi \in \mathbb{R}, \phi > -1$,

$$T_{\phi}^{-1}(x, y) = \begin{bmatrix} \frac{1}{1+\phi} & 0 \\ 0 & \frac{1}{1+\phi} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x/(1+\phi) \\ y/(1+\phi) \end{bmatrix} \tag{8}$$

3.8. Pixel-wise Transformations

Pixel-wise transformations are defined by the cumulative effects of brightness and contrast (i.e., simply changes in lighting) acting on a pixel $X_{i,j}$ of an image X where the respective brightness and contrast parameters are $c, b \in \mathbb{R}$. These transformations are defined by Balunovic et al. [15] and Mohapatra et al. [14] as,

$$X_{i,j}^{'} = \min(1, \max(0, (1 + c) \cdot X_{i,j} + b)) \tag{9}$$

In our work, the pixel-wise transformations are parameterized by darkening and brightening offsets, $\zeta_L, \zeta_H \in \mathbb{R}$, and the upper and lower bounds for the perturbations are specified as,

$$X_{i,j}^{'} = (X_{i,j} - \zeta_L \ , \ X_{i,j} + \zeta_H) \tag{10}$$

3.9. Composite Geometric Perturbations

Composite geometric perturbations based on real-world phenomena are a combination of one affine transformation (like rotation, translation or scaling) and a pixel-wise transformation (like changes in lighting). These perturbations are visually represented in figure 2 as,

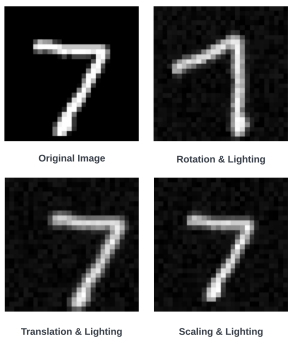

Figure 2: Composite geometric perturbations based on real-world phenomena

4. Proposed Method

In this section, we present a method to capture composite geometric perturbations based on real-world phenomena (like the combined effects of axial rotations and changes in lighting) in the form of abstract intervals. The basic idea is to pass these abstract intervals to the VoTE property checker [3] to verify stability and compute the extended stability metrics, which will be explained in this section.

4.1. Abstracting the Perturbations

For a particular input pixel x, if s is the total number of geometric transformation (eg. rotation) steps for a given transformation, there exists a function, $\lambda(x)$ that projects the effects of the transformation on this pixel at each step

$$\lambda(x) = \{x_1, \ ..., \ x_s\} \tag{11}$$

such that x_i is the outcome of the transformation at each step i, $1 \leq i \leq s$. In order to capture these pixel values in the interval domain, we use the abstraction function α, which is designed to compute an over-approximation of the problem,

$$\alpha(\lambda(x)) = (min \ \lambda(x), \ max \ \lambda(x)) \tag{12}$$

For a *composite* geometric transformation involving a combination of one affine transformation (rotation, scaling or translation) along with changes in lighting conditions where ζ_L and ζ_H are real-world darkening and brightening margins, if p represents the initial pixel *value* before the transformations, the above (single transformation) abstraction function can be redefined as,

$$\alpha(\lambda(x)) = ((min \ \lambda(x) - \zeta_L) - p, \ (max \ \lambda(x) + \zeta_H) - p) \tag{13}$$

The redefined abstraction function returns a tuple (ϵ_L, ϵ_H) that represents the perturbation margins of the pixel where,

$$\epsilon_L = (min \ \lambda(x) - \zeta_L) - p$$
$$\epsilon_H = (max \ \lambda(x) + \zeta_H) - p$$

The lower and upper perturbation bounds are computed by applying the perturbation margins (ϵ_L, ϵ_H) to the pixel's intensity as $(p + \epsilon_L, \ p + \epsilon_H)$

Running Example

If a pixel x in an image with an initial intensity, $p = 5$ is rotated between 0° and 5° in 5 steps, from equation (11),

$$\lambda(x) = \{6, \ 2, \ 3, \ 1, \ 4\}$$

Applying equation (12),

$$\alpha(\lambda(x)) = (1, \ 6)$$

If $\zeta_L = \zeta_H = 1$, applying equation (13),

$$\alpha(\lambda(x)) = ((1-1)-5,\ (6+1)-5)$$

$$(\epsilon_L,\ \epsilon_H) = (-5,\ 2)$$

These pixel-level perturbation margins are then applied to the initial pixel value before the transformations to get the pixel-level perturbation bounds that are representative of composite geometric perturbations (like rotations and changes in lighting).

$$(p+\epsilon_L,\ p+\epsilon_H) = (5+(-5),\ 5+2) = (0,\ 7)$$

From this example, we see that the pixel x can take on any value between (0, 7) during a composite geometric transformation.

4.2. Feature Space Modelling

The abstraction process helps determine the upper and lower pixel-level perturbation margins in order to achieve efficient computations during the verification process. Since VoTE operates based on abstracting each point with an interval in one dimension, when applying to a multidimensional space we need to extend the abstraction function through feature space modelling to simulate these perturbations for vision-based systems.

Algorithm 1: Feature Space Modelling: Rotation

Input: Angle 1 (n_1), Angle 2 (n_2), Offset-High (ζ_H), Rot. Steps (r), Offset-Low (ζ_L), Images (I)
Output: Perturbation Definitions (τ)

$N, \tau \leftarrow \emptyset, ( )$ ▷ N = Set of angles to apply
$N \leftarrow$ *generate-set-of-angles* $(n_1,\ n_2,\ r)$
for *each image*, $i \in I$ **do**
  $\Delta_i \leftarrow ( )$ ▷ Image level perturbations
  for *each pixel*, $x \in i$ **do**
    $\lambda \leftarrow \emptyset$ ▷ Transformed pixel values
    for *each angle*, $v \in N$ **do**
      $x_v \leftarrow$ rotate-image (i, v)
      $\lambda \leftarrow \lambda \cup \{x_v\}$
    $(\epsilon_L,\ \epsilon_H) \leftarrow \alpha(\lambda)$ ▷ α from equation (13)
    append $(\epsilon_L,\ \epsilon_H)$ to Δ_i
  append Δ_i to τ

Note that while Algorithm 1 contains axial rotations as the affine transformation, this can be easily adapted to the corresponding other affine transformations, i.e., scaling or translation computed by similar respective algorithms analogously.

4.3. Verification Workflow

In this section, we present the verification workflow that shows the feature space modelling process incorporated into the VoTE [3] framework. We will use one of the case studies in section 5.1 to illustrate the flow in Figure 3. If we consider rotations between 0° and 10°, I_L (which is the image generated by applying ϵ_L associated with all pixels in a particular image) and I_H (the corresponding image generated by applying ϵ_H) represent the first and last images in the verification process. We then ask the VoTE Property Checker to verify all the images in the transformed series.

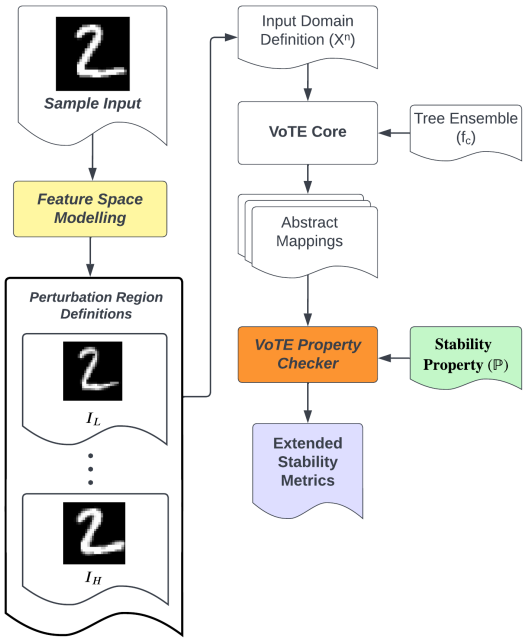

Figure 3: Verification Workflow (MNIST)

The property checker returns a *pass* when the classifier makes stable predictions on all the images in the transformed series. If the result of the verification process is *fail*, the property does not hold (the classifier is not stable). In this case, the VoTE property checker can be configured to return a counterexample that can help with further analysis. The illustration of the approach in more detail is described with use cases in section 5.

4.4. Classifier Correctness

The correctness of a classifier (f_c), with respect to an input $\bar{x}$, denoted by $IsCorrect(f_c, \bar{x})$ is proven when f_c predicts the correct label (l) for an input sample ($\bar{x}$) according to a ground truth. This notion of **correctness** computed over a test set (X_{test}) can be used to quantify accuracy as

$$\textbf{accuracy} = \frac{|\{\ \overline{x} \in X_{test} \mid IsCorrect(f_c\ ,\ \overline{x})\}|}{|X_{test}|} \tag{14}$$

4.5. Safety Property: Classifier Stability

Accuracy on a test set is the most commonly used metric to evaluate a classification model. However, accuracy alone is not enough for convincing safety arguments in models that use machine learning. In this paper, we consider Stability, a property commonly used in related works. Robustness (and other metrics consistent with the literature) are generalised to a notion of classifier stability and correctness [9]. Note that compliance with stability and its associated metrics alone isn't generally enough to ensure system safety. Safety Engineers typically define requirements on software functions that are much richer than this property alone.

Let $f_c : \overline{x} \rightarrow y$ be the classifier subject to verification where $\overline{x}$ (the image) is an n-array vector consisting of individual pixels, and y is the output class or label. $\epsilon_L\ ,\ \epsilon_H \in \mathbb{R}$ are lower and upper perturbation margins ($\epsilon_L < \epsilon_H$) associated with an individual pixel in the image. If n is the number of pixels in the image, $\Delta = ((\epsilon_{L_1},\ \epsilon_{H_1}),\ ...,\ (\epsilon_{L_n},\ \epsilon_{H_n}))$ is a sequence containing tuples of pixel-level perturbation margins associated with the image. $\overline{\omega}$ is an n-tuple of perturbations, with $\overline{\omega} = (\omega_1,\ ...,\ \omega_n)$ and $\epsilon_{L_i} \leqslant \omega_i \leqslant \epsilon_{H_i}$. If we consider a test sample $\overline{x}$ and all possible variations of $\overline{\omega}$, the classifier f_c is *stable* on that sample (denoted by $IsStable(f_c\ ,\ \overline{x})$) if and only if

$$f_c(\overline{x}) = f_c(\overline{x}')\ ,\ where\ \overline{x}' = \overline{x} + \overline{\omega} \tag{15}$$

The above expression can easily be extended over a test set (X_{test}). Note that $\overline{\omega}$ when added to the base image ($\overline{x}$) produces a version of that base image that represents the combined effects of a geometric transformation and changes in lighting. These different versions of the base image can then be analyzed by the verification engine.

4.6. Extended Stability Metrics

Stability and Robustness are standard metrics commonly used in the literature on adversarial machine learning. They are generally used to quantify the security of classifiers at test time (for e.g., evasion attacks). Beyond stability, Ranzato and Zanella [9] define four additional metrics - robustness (R), fragility (F), vulnerability (V) and breakage (B) that combine classifier stability with prediction accuracy in the presence of input perturbations. We recall

$$\mathbf{R} = \frac{|\{\ \overline{x} \in X_{test} \mid IsCorrect(f_c\ ,\ \overline{x}) \wedge IsStable(f_c\ ,\ \overline{x})\ \}|}{|X_{test}|}$$

$$\mathbf{F} = \frac{|\{\ \overline{x} \in X_{test} \mid IsCorrect(f_c\ ,\ \overline{x}) \wedge \neg IsStable(f_c\ ,\ \overline{x})\ \}|}{|X_{test}|}$$

$$\mathbf{V} = \frac{|\{\ \overline{x} \in X_{test} \mid \neg IsCorrect(f_c\ ,\ \overline{x}) \wedge IsStable(f_c\ ,\ \overline{x})\ \}|}{|X_{test}|}$$

$$\mathbf{B} = \frac{|\{\ \overline{x} \in X_{test} \mid \neg IsCorrect(f_c\ ,\ \overline{x}) \wedge \neg IsStable(f_c\ ,\ \overline{x})\ \}|}{|X_{test}|}$$

We incorporate these definitions into VoTE's Property Checker which formally verifies classifier stability and automatically computes these metrics at the same time.

4.7. VoTE Property Checker

Recall that from the preliminaries, the VoTE property checker (*VoTE-Pc*) returns a pass if the property is satisfied. For a tree-ensemble model (f_c), and a test sample $\overline{x}$ ($\overline{x} \in X_{test}$), the following equations are used to verify stability and correctness, which can be extended to compute the metrics from the equations in section 4.6.

$$\textbf{\textit{is\_correct}} \leftarrow \textit{VoTE-Pc}\ (\overline{x}\ ,\ IsCorrect(f_c\ ,\ \overline{x})) \tag{16}$$

$$\textbf{\textit{is\_stable}} \leftarrow \textit{VoTE-Pc}\ (\overline{x}\ ,\ IsStable(f_c\ ,\ \overline{x})) \tag{17}$$

5. Case Studies

We apply our three affine transformations combined with changes in lighting to two case studies. We verify stability and compute the extended metrics associated with classifier stability and correctness. Due to space restrictions, we will only show the outcomes of composite rotations in one study and composite scaling in the other. We use scikit-learn [6] to train random forests and CatBoost [7] to train gradient boosting machines. Experiments are conducted on a Windows 11 Machine running Ubuntu 20.04 in WSL Mode (Windows Subsystem for Linux). The machine comes equipped with an Intel Core i7-10875H CPU and 16 GB RAM.

5.1. Digit Recognition

The MNIST dataset [22] is a collection of hand-written digits commonly used to evaluate machine learning algorithms. The dataset contains 70,000 gray scale images with a resolution of 28x28 pixels at 8bpp, encoded as a tuple of 784 pixels. The input regions surrounding each sample in the test set were defined using feature space modelling (section 4.3). The parameters for rotation were chosen as follows: Rotation Range = $0 - 10°$, Rotation Steps = 1001. For the changes in lighting, the parameters were chosen as: $\zeta_L = 2$ (darkening) and $\zeta_H = 3$ (brightening). To make this case study even more realistic, the training data was augmented (doubled to 120,000 training samples) by adding samples that were randomly rotated

in the range of ±10° using Keras [24]. The verification process was carried out on 10,000 test samples.

Param.		A (%)		R (%)		F (%)		B (%)		T (s)	
d	B	RF	GB	RF	GB	RF	GB	RF	GB	RF	GB
3	3	51.26	61.75	14.36	13.61	36.90	48.14	48.74	38.25	0.21	0.40
3	5	57.26	71.60	7.03	15.81	50.23	55.82	42.74	28.37	0.22	0.26
5	5	76.59	79.56	2.75	40.89	73.84	38.67	23.41	20.44	0.22	0.26
5	10	80.23	87.16	0.69	31.00	79.54	56.16	19.77	12.84	0.23	0.56
5	15	81.95	89.94	0.88	23.74	81.07	66.20	18.05	10.06	0.36	13.33
5	20	83.20	91.61	1.37	20.21	81.83	71.40	16.80	8.39	1.81	168.12

Table 1: Results (Combined Rotations and Lighting)

Table 1 lists random forests (RF) and gradient boosting machines (GB) included in the experiments with their maximum tree depth (d), number of trees (B), accuracy (A), the extended stability metrics - robustness (R), fragility (F), and breakage (B) along with the elapsed time (T) taken for verification (in seconds). The column for vulnerability was omitted as it was zero in all experiments.

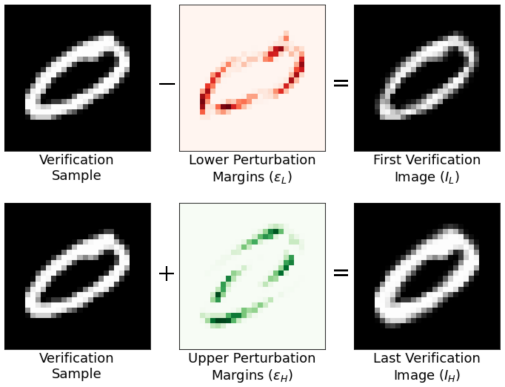

Figure 4: Crafting Perturbations (Composite Rotations)

Figure 4 shows the construction of the first and last images in the verification series for a particular sample. From this figure, it is visible that the two images, i.e., the first and the last resemble the ground truth, and the rest of the images in the series are similar in that sense.

5.2. Handwritten Character Recognition

In our next experiments, we use the EMNIST-Letters dataset, which is a variant of the EMNIST Dataset [23] that includes upper and lower case handwritten characters. This dataset contains 145,600 samples across 26 balanced classes. Each sample has a resolution of 28x28 pixels at 8bpp, encoded as a tuple of 784 pixels. The input regions surrounding each sample in the test set were defined using feature space modelling. The parameters for scaling were chosen as follows: Scaling Steps = 201, Scale Factor = ±10% which represents a 10% Zoom (in and out). For the changes in lighting, the parameters were chosen as: $\zeta_L = 2$ (darkening) and $\zeta_H = 3$ (brightening). To make this case study even more realistic, the training data was augmented (doubled to 249,600 training samples) by adding samples that were randomly scaled in the range of ±10% using Keras [24]. The verification process was carried out on 10,000 test samples.

Param.		A (%)		R (%)		F (%)		B (%)		T (s)	
d	B	RF	GB	RF	GB	RF	GB	RF	GB	RF	GB
3	3	30.85	36.44	2.34	1.55	28.51	34.89	69.15	65.36	0.22	0.19
3	5	35.39	43.92	1.72	1.43	33.67	42.49	64.61	56.08	0.24	0.22
5	5	45.83	50.75	3.46	2.23	42.37	48.52	54.17	49.25	0.23	0.26
5	10	51.27	61.83	2.99	0.68	48.28	61.15	48.73	38.17	0.33	0.79
5	15	53.42	68.04	2.83	0.13	50.59	67.91	46.58	31.96	2.01	21.50
5	20	54.41	71.56	3.51	0.10	50.90	71.46	45.59	28.44	6.10	264.69

Table 2: Results (Combined Scaling and Lighting)

Table 2 lists random forests (RF) and gradient boosting machines (GB) included in the experiments with their maximum tree depth (d), number of trees (B), accuracy (A), the extended stability metrics - robustness (R), fragility (F), and breakage (B) along with the elapsed time (T) taken for verification (in seconds). Again, the column for vulnerability was omitted as it was zero in all experiments.

In regard to this case study, if we consider the EMNIST [23] letter *a* and scaling in the range of ±10 %, with $\zeta_L = 2$ and $\zeta_H = 3$, I_L and I_H represent the first and last images in the verification process. We then ask the VoTE Property Checker to verify all the images in the transformed series. Figure 5 shows a fragment of the verification flow.

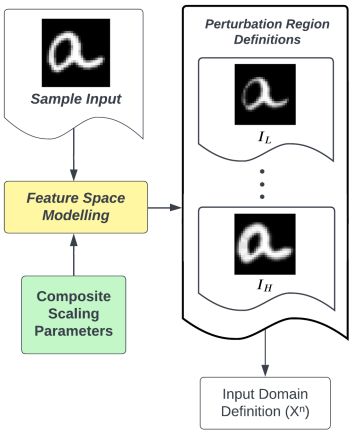

Figure 5: Verification Workflow Fragment (EMNIST)

Discussions

From the experimental results in Table 1 and Table 2, we note that the robustness of the learned models with

respect to composite geometric perturbations is much lower compared to the standard model accuracy (which is somewhat expected). In the first case study, gradient boosting machines performed slightly better than random forests in terms of robustness. Increasing the number of trees improves accuracy but causes a drop in robustness. This indicates that the models were overfitted and that adding compositely perturbed images to the training set may improve robustness. The high numbers for fragility indicated that while these samples were identified correctly, the classifier was unable to make stable predictions on them in presence of these types of perturbations. We would also like to mention that the verification process would require a domain expert, due to the complex nature of these perturbations. For instance, if you pick a large rotation range (or a large scaling or translation range), there will be a significant number of images in the verification series that won't resemble the ground truth label. In this case, the verification process needs to be done in small ranges.

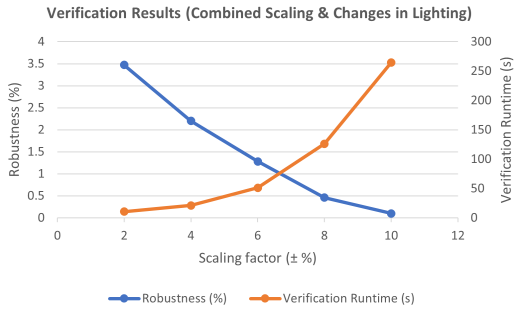

Figure 6: Comparing Verification Metrics

We repeated the experiments in the second case study by varying the scale factors as: ±2%, ±4%, ±6%, ±8%, ±10% for the largest Gradient Boosting Machine. In figure 6, we can see that as the scale factor is increased, the robustness of the model decreases and the verification runtime increases. This is expected as the number of perturbed pixels increases as the scale factor is increased.

Also, while the tree ensemble models chosen for the experiments could be considered trivial, we would like to point out that the verification problem was certainly non-trivial considering the composite nature of these perturbations, and the fact that the models were trained on high-dimensional data. What is positive in this context is the fact that performing such analyses in tractable time is at all possible given the enormous search space for these perturbations.

Considering the complexity of the verification problem, it is impressive that VoTE was able to efficiently verify these models within a fraction of a second for most of the experiments. For larger models, while the verification time does increase, this issue is resolvable since due to the memory-efficient structure of the VoTE core algorithm, computations can always be parallelised (not used in our experiments).

VoTE captures multiple inputs (precisely) in the interval domain, but whenever a prediction model violates the stability property, the property checker returns a counterexample which lies in the violating region. Figure 7 represents a few of the many counterexamples discovered by VoTE during the verification process (MNIST-Digits). Since the perturbations are almost indistinguishable by the naked eye, they are highlighted in pink. These counterexamples can be used for debugging or further analysis to repair or retrofit the models iteratively until provable stability is eventually achieved. For instance, counterexamples used for guided model training can be embedded into a loss function to achieve better stability.

Figure 7: Counterexamples

6. Conclusions and Future Works

Applying formal verification to show that a system exhibits its intended behaviour can help increase the trustworthiness of the decisions made by intelligent systems. In this work, we show that tree-ensemble based models with comparatively good prediction accuracies perform spectacularly poorly when presented with samples containing composite geometric perturbations based on real-world phenomena.

Apart from the profound impacts on safety, these types of perturbations can also violate the security properties of classifiers which could be detrimental in safety-critical scenarios. To address this issue, the parameters in our method can be tweaked to simultaneously check for adversarial perturbations in addition to composite geometric perturbations. This represents a step towards safe and secure AI. Finally, our method can also be used with different types of models and verification engines depending on the context.

For future works, we plan on exploring different abstract domains which can be used to tightly capture these perturbations to increase the scalability of VoTE to

even higher-dimensional inputs. We also plan to explore different tree selection strategies, i.e., a systematic analysis of the order in which the trees in the ensemble are analyzed in VoTE's abstraction-refinement pipeline to further enhance the efficiency of the verification process against the large nature of these perturbations.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] Breiman, L. (2004). Random Forests. Machine Learning, 45, 5-32.

[2] Friedman, J.H. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29, 1189-1232.

[3] Törnblom, J., & Nadjm-Tehrani, S. (2019). An Abstraction-Refinement Approach to Formal Verification of Tree Ensembles. SAFECOMP Workshops.

[4] Irsoy, O., Yildiz, O.T., & Alpaydin, E. (2012). Soft decision trees. Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), 1819-1822.

[5] Wing, J. M. (2021). Trustworthy AI. In Communications of the ACM (Vol. 64, Issue 10, pp. 64–71). Association for Computing Machinery (ACM). https://doi.org/10.1145/3448248

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Louppe, G., Prettenhofer, P., Weiss, R., Weiss, R.J., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res., 12, 2825-2830.

[7] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. Advances in neural information processing systems, 31.

[8] Gokhale, T., Anirudh, R., Kailkhura, B., Thiagarajan, J.J., Baral, C., & Yang, Y. (2021). Attribute-Guided Adversarial Training for Robustness to Natural Perturbations. AAAI.

[9] Ranzato, F., & Zanella, M. (2020). Abstract Interpretation of Decision Tree Ensemble Classifiers. AAAI.

[10] Scher, S., & Trügler, A. (2022). Robustness of Machine Learning Models Beyond Adversarial Attacks. ArXiv, abs/2204.10046.

[11] Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., & Yi, X. (2020). A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev., 37, 100270.

[12] Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., & Madry, A. (2019). Exploring the Landscape of Spatial Robustness. ICML.

[13] Pei, K., Cao, Y., Yang, J., & Jana, S.S. (2019). Towards Practical Verification of Machine Learning: The Case of Computer Vision Systems. ICSE Workshop on Testing for Deep Learning and Deep Learning for Testing

[14] Mohapatra, J., Weng, T., Chen, P., Liu, S., & Daniel, L. (2020). Towards Verifying Robustness of Neural Networks Against A Family of Semantic Perturbations. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 241-249.

[15] Balunovic, M., Baader, M., Singh, G., Gehr, T., & Vechev, M.T. (2019). Certifying Geometric Robustness of Neural Networks. NeurIPS.

[16] Wu, H., Tagomori, T., Robey, A., Yang, F., Matni, N., Pappas, G., Hassani, H., Pasareanu, C.S., & Barrett, C. (2022). Toward Certified Robustness Against Real-World Distribution Shifts. ArXiv, abs/2206.03669.

[17] Yang, R., Laurel, J., Misailovic, S., & Singh, G. (2022). Provable Defense Against Geometric Transformations. arXiv preprint arXiv:2207.11177.

[18] Katz, G., Barrett, C.W., Dill, D.L., Julian, K.D., & Kochenderfer, M.J. (2017). Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. International Conference on Computer Aided Verification.

[19] Gowal, S., Qin, C., Huang, P., cemgil, T., Dvijotham, K., Mann, T.A., & Kohli, P. (2020). Achieving Robustness in the Wild via Adversarial Mixing With Disentangled Representations. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 1208-1217.

[20] Hong, T., Hu, M., Yin, T., & Wang, S. (2022). A Multi-Scale Convolutional Neural Network for Rotation-Invariant Recognition. Electronics.

[21] Peri, A., Mehta, K., Mishra, A., Milford, M., Garg, S., & Krishna, K.M. (2022). ReF - Rotation Equivariant Features for Local Feature Matching. ArXiv, abs/2203.05206.

[22] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proc. IEEE, 86, 2278-2324.

[23] Cohen, G., Afshar, S., Tapson, J.C., & Schaik, A.V. (2017). EMNIST: an extension of MNIST to handwritten letters. ArXiv, abs/1702.05373.

[24] Chollet, F. (2018). Keras: The Python Deep Learning library. https://keras.io

[25] Cleland, G. M., Sujan, M. A., Habli, I., & Medhurst, J. (2012). Evidence: using safety cases in industry and healthcare. The Health Foundation.

[26] Törnblom, J., & Nadjm-Tehrani, S. (2021). Scaling up Memory-Efficient Formal Verification Tools for Tree Ensembles. ArXiv, abs/2105.02595.