FairQuant: Certifying and Quantifying Fairness of Deep Neural Networks

Brian Hyeongseok Kim
University of Southern California
Los Angeles, USA

Jingbo Wang
Purdue University
West Lafayette, USA

Chao Wang
University of Southern California
Los Angeles, USA

***Abstract*—We propose a method for formally certifying and quantifying *individual fairness* of deep neural networks (DNN). Individual fairness guarantees that any two individuals who are identical except for a legally protected attribute (e.g., gender or race) receive the same treatment. While there are existing techniques that provide such a guarantee, they tend to suffer from lack of scalability or accuracy as the size and input dimension of the DNN increase. Our method overcomes this limitation by applying abstraction to a symbolic interval based analysis of the DNN followed by iterative refinement guided by the fairness property. Furthermore, our method lifts the symbolic interval based analysis from conventional *qualitative* certification to *quantitative* certification, by computing the percentage of individuals whose classification outputs are provably fair, instead of merely deciding if the DNN is fair. We have implemented our method and evaluated it on deep neural networks trained on four popular fairness research datasets. The experimental results show that our method is not only more accurate than state-of-the-art techniques but also several orders-of-magnitude faster.**

I. Introduction

The problem of certifying the fairness of machine learning models is more important than ever due to strong interest in applying machine learning to automated decision making in various fields from banking [1] and healthcare [2] to public policy [3] and criminal justice [4]. Since the decisions are socially sensitive, it is important to certify that the machine learning model indeed treats individuals or groups of individuals fairly. However, this is challenging when the model is a deep neural network (DNN) with a large number of hidden parameters and complex nonlinear activations. The challenge is also exacerbated as the network size and input dimension increase. In this work, we aim to overcome the challenge by leveraging abstract interpretation techniques to certify fairness both qualitatively and quantitatively.

Our work focuses on *individual fairness* which, at a high level, requires that similar individuals are treated similarly [5]. Here, *similar individuals* are those who differ only in some legally *protected* input attribute (e.g., gender or race) but agree in the *unprotected* attributes, and *being treated similarly* means that the DNN generates the same classification output.[1] Let the DNN be a function $f : X \to Y$ from input domain X to output range Y, where an individual $x \in X$ is an input and a class label $y \in Y$ is the output. Assume that each input $x = \langle x_1, \ldots, x_D \rangle$ is a D-dimensional vector, and x_j, where $1 \leq j \leq D$, is a *protected* attribute. We say that the DNN is *provably fair* (certified) for the entire input domain X if $f(x) = f(x')$ holds for any two individuals $x \in X$ and $x' \in X$ that differ only in x_j but agree in the unprotected attributes ($\forall x_i$ where $i \neq j$). Conversely, the DNN is *provably unfair* (falsified) for input domain X if $f(x) \neq f(x')$ holds for any two individuals ($x \in X$ and $x' \in X$) that differ only in the protected attribute. If the DNN is neither certified nor falsified, it remains *undecided*.

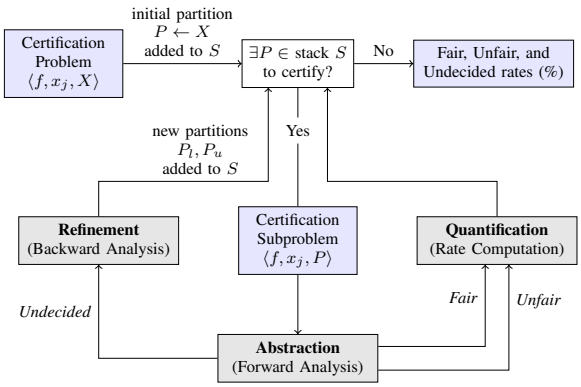

Fig. 1. FairQuant for certifying and quantifying fairness of a DNN model f where x_j is a protected attribute and X is the input domain.

Given a DNN f, a protected attribute x_j, and an input domain X, a *qualitative* certification procedure aims to determine whether f is *fair*, *unfair*, or *undecided* for all $x \in X$. Qualitative analysis is practically important because, if f is provably fair, the model may be used *as is*, but if f is provably unfair, the model should not be used to make decision for any $x \in X$. When the result of qualitative analysis is *undecided*, however, there is a need for *quantitative* analysis, to compute the degree of fairness. For example, the degree of fairness may be measured by the percentage of individuals in input domain X whose classification outputs are provably fair.

Both *qualitative* certification and *quantitative* certification are hard problems for deep neural networks. While there are many verification tools for deep neural networks, existing verifiers such as ReluVal [7], DeepPoly [8], and α-β-CROWN [9] focus on certifying perturbation robustness, which is a fundamentally different property, and cannot certify

[1]This notion can be understood as causal fairness [35] or dependency fairness [31], which is a non-probabilistic form of counterfactual fairness [6].

individual fairness. To the best of our knowledge, the only existing technique for certifying individual fairness of a DNN is Fairify [10]. However, since it directly analyzes the behavior of a DNN in the concrete domain using the SMT solver, the computational cost is extremely high; as a result, Fairify can only certify tiny networks. Furthermore, it cannot quantify the degree of fairness. Prior works on quantitative analysis of fairness focus on either testing [11], [12], [13], which do not lead to sound certification, or statistical parity [14], [15] that concerns another type of fairness, group fairness, which differs significantly from individual fairness.

To fill the gap, we propose the first scalable method for *certifying* and *quantifying* individual fairness of a DNN. Our method, named FairQuant, takes a certification problem (consisting of the DNN f, protected attribute x_j, and input domain X) as input and returns one of the following three outputs: (1) *certified (fair)* for all input $x \in X$; (2) *falsified (unfair)* for all input $x \in X$; or (3) *undecided*, meaning that f is neither 100% *fair* nor 100% *unfair*. In the third case, our method also computes the percentage of inputs in X whose classification outputs are provably *fair*. More specifically, our method provides a *lower bound* of the certified percentage, which can guarantee that the DNN meets a certain requirement, e.g., the DNN is individually fair for at least 80% of all inputs in X.

As shown in Fig. 1, FairQuant iterates through three steps: abstraction (forward analysis), refinement (backward analysis), and quantification (rate computation). Assuming that the legally protected attribute x_j has two possible values (e.g., male and female), forward analysis tries to prove that, for each x in the input partition P (which is the entire input domain X initially), flipping the value of the protected attribute of x does not change the model's output. This is accomplished by propagating two symbolic input intervals I ($\forall x \in P$ that are male) and I' ($\forall x' \in P$ that are female) to compute the two corresponding output intervals that are overapproximated. If the classification labels (for all x and x') are the same, our method returns *certified (fair)*. On the other hand, if the classification labels (for all x and x') are different, our method returns *falsified (unfair)*. In these two cases, 100% of the inputs in the partition P are resolved.

Otherwise, we perform refinement (backward analysis) by splitting P into partitions P_l and P_u and apply forward analysis to each of these new partitions. Since smaller partitions often lead to smaller approximation errors, refinement has the potential to increase the number of certified (or falsified) inputs and decrease in the number of undecided inputs. To ensure that our method terminates quickly, we propose two *early termination* conditions based on the refinement depth of each partition $P \subseteq X$. The refinement depth is the number of times X is partitioned to produce P. There are two predefined thresholds. Once the refinement depth exceeds the higher threshold, we classify the partition P as *undecided* and avoid splitting it further. But if the refinement depth exceeds the lower threshold without exceeding the higher threshold, we use random sampling to try to find a concrete example $x \in P$ that violates the fairness property. If such a counterexample is found, we classify P as *undecided* and avoid splitting it further. Otherwise, we keep splitting P into smaller partitions.

We have evaluated our method on a large number of deep neural networks trained using four widely-used datasets for fairness research: Bank [16] (for predicting marketing), German [17] (for predicting credit risk), Adult [18] (for predicting earning power), and Compas [19] (for predicting recidivism risk). For comparison, we apply Fairify [10] since it represents the current state-of-the-art in certifying individual fairness; we also apply α-β-CROWN since it is currently the best robustness verifier for deep neural networks. Our results show that α-β-CROWN is not effective in certifying individual fairness. As for Fairify, our method FairQuant significantly outperforms Fairify in terms of both accuracy and speed for all DNN benchmarks. In fact, FairQuant often completes certification in seconds, whereas Fairify often times out after 30 minutes and certifies nothing or only a tiny fraction of the entire input domain.

To summarize, this paper makes the following contributions:

- We propose the first scalable method for certifying and quantifying individual fairness of DNNs using symbolic interval based analysis techniques.
  1) For forward analysis, we propose techniques for more accurately deciding if the DNN is fair/unfair for all inputs in an input partition.
  2) For refinement, we propose techniques for more effectively deciding how to split the input partition.
  3) For quantification, we propose techniques for efficiently computing the percentages of inputs whose outputs can be certified and falsified.
- We demonstrate the advantages of our method over the current state-of-the-art on a large number of DNNs trained using four popular fairness research datasets.

The remainder of this paper is organized as follows. First, we motivate our work in Section II using examples. Then, we present the technical background in Section III. Next, we present the high-level procedure of our method in Section IV, followed by detailed algorithms of the abstraction, refinement, and quantification subroutines in Sections V, VI and VII. We present the experimental results in Section VIII, review the related work in Section IX, and finally give our conclusions in Section X.

II. Motivation

In this section, we use an example to illustrate the limitations of existing methods.

A. The Motivating Example

Fig. 2 (left) shows a DNN for making hiring decisions. It has three input nodes (i_1, i_2 and i_3), two hidden neurons (h_1 and h_2) and one output node (o). The values of h_1 and h_2 are computed in two steps: first, the values of i_1, i_2 and i_3 are multiplied by the edge weights before they are added up; then, the result is fed to an activation function. For instance,

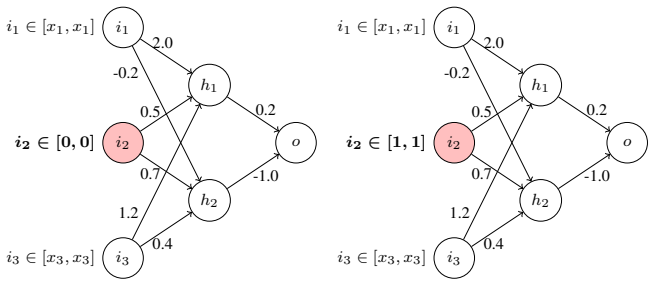

Fig. 2. Symbolic interval analysis of an example DNN for making hiring decisions: the left figure is for female applicants ($i_2 \in [0, 0]$), and the right figure is for male applicants (where $i_2 \in [1, 1]$). Except for the protected attribute i_2, the symbolic intervals of the other attributes are the same.

the activation function may be ReLU$(z) = max(0, z)$. The output of the entire network f is based on whether the value of o is above 0; that is, positive label is generated if $o > 0$; otherwise, negative label is generated.

The DNN takes an input vector x with three attributes: x_1 is the interview score of the job applicant, x_2 is the gender (0 for female and 1 for male), and x_3 is the number of years of experience. Furthermore, x_2 is the protected attribute while x_1 and x_3 are unprotected attributes. In general, the input domain may be unbounded, e.g., when some attributes are real-valued variables. However, for illustration purposes, we assume that the input domain is $X = \{x \mid x_1 \in \{1, 2, 3, 4, 5\}, x_2 \in \{0, 1\},$ and $x_3 \in \{0, 1, 2, 3, 4, 5\}\}$, meaning that X has a total of $5 \times 2 \times 6 = 60$ individuals.

Consider the individual $x = \langle 5, 0, 5\rangle$, meaning that $x_1 = 5$, $x_2 = 0$ and $x_3 = 5$. According to the DNN in Fig. 2, the output is the positive label. After flipping the value of the protected attribute x_2 from 0 to 1, we have the individual $x' = \langle 5, 1, 5\rangle$, for which the DNN's output is also the positive label. Since the DNN's output is oblivious to the gender attribute, we say that it is fair for this input x.

Consider another individual $x = \langle 1, 0, 5\rangle$ whose gender-flipped counterpart is $x' = \langle 1, 1, 5\rangle$. Since the DNN produces the negative label as output for both, it is still fair for this input x.

To summarize, the DNN f may be fair regardless of whether a particular $x \in X$ receives a positive or negative output; as long as x receives the same label as its counterpart x', the DNN is considered fair.

In contrast, since the individual $x = \langle 1, 0, 3\rangle$ and its counterpart $x' = \langle 1, 1, 3\rangle$ receive different outputs from f, where x gets the positive label but x' gets the negative label, the DNN is not fair for this input x. Furthermore, this pair (x, x') serves as a counterexample.

B. Limitations of Prior Work

One possible solution to the fairness certification problem as defined above would be explicit enumeration of the (x, x') pairs. For each $x \in X$, we may flip its protected attribute to generate x' and then check if $f(x) = f(x')$. However, since the size of the input domain X may be extremely large or infinite, this method would be prohibitively expensive in practice.

Another possible solution is to leverage existing DNN robustness verifiers, such as ReluVal [7], [20], DeepPoly [8], and α-β-CROWN [9]. However, since robustness and individual fairness are fundamentally different properties, applying a robustness verifier would not work well in practice. The reason is because a robustness verifier takes an individual x and tries to prove that small perturbation of x (often defined by $||x - x'|| < \delta$, where δ is a small constant) does not change the output label. However, during fairness certification, we are not given a concrete individual x; instead, we are supposed to check for all $x \in X$ and $x' \in X$, where $x_j \neq x'_j$. If we force a robustness verifier to take a symbolic input I ($\forall x \in X$), it would try to prove that the DNN produces the same output label for all inputs in X (implying that the DNN makes the same decision for all inputs in X).

Recall our example network f in Fig. 2. While our method can prove that f is fair for an input domain that contains $x = \langle 5, 0, 5\rangle$ and $x' = \langle 5, 1, 5\rangle$ (both receive a positive outcome) as well as $x = \langle 1, 0, 5\rangle$ and $x' = \langle 1, 1, 5\rangle$ (both receive a negative outcome), this cannot be accomplished by a robustness verifier (since it is almost never possible for all individuals in the input domain to have the same outcome).

The only currently available method for (qualitatively) certifying individual fairness of a DNN is Fairify [10], which relies on the SMT solver and may return one of the following results: SAT (meaning that there exists a counterexample that violates the fairness property), UNSAT (meaning that there is no counterexample), or UNKNOWN (meaning that the result remains inconclusive). The main problem of Fairify is that it works directly in the concrete domain by precisely encoding the non-linear computations inside the DNN as logical formulas and solving these formulas using the SMT solver. Since each call to the SMT solver is NP-complete, the overall computational cost is high. Although Fairify attempts to reduce the computational cost by partitioning input domain *a priori* and heuristically pruning logical constraints, it does not scale as the network size and input dimension increase. Indeed, our experimental evaluation of Fairify shows that only tiny networks (with ≤ 100 neurons) can be certified.

C. Novelty of Our Method

We overcome the aforementioned (accuracy and scalability) limitations by developing a method that is both *scalable* and able to *quantify* the degree of fairness.

First, FairQuant relies on abstraction to improve efficiency/scalability while maintaining the accuracy of symbolic forward analysis. This increases the chance of quickly certifying more input regions as fair or falsifying them as unfair, and decreases the chance of leaving them as undecided. Specifically, we use symbolic interval analysis, instead of the SMT solver used by Fairify. The advantage is that symbolic interval analysis focuses on the behavior of the DNN in an abstract domain, which is inherently more efficient and scalable than analysis in the concrete domain.

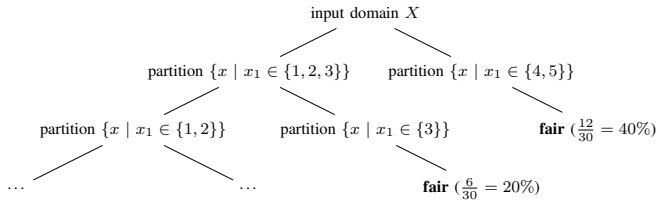

Fig. 3. Iterative refinement tree for the example DNN in Fig. 2, to increase the chance of certifying or falsifying the DNN within an input partition.

Second, FairQuant relies on iterative refinement (partitioning of the input domain) to improve the accuracy of forward analysis. Instead of creating input partitions *a priori*, it conducts iterative refinement on a "need-to" basis guided by the fairness property to be certified. This makes it more effective than the static partitioning technique of Fairify, which divides the input domain into a fixed number of equal chunks even before verifying any of them.

To see why iterative refinement can improve accuracy, consider our running example in Fig. 2. Initially, forward analysis is applied to the DNN in the entire input domain X, for which the certification result is *undecided*. During refinement, our method would choose x_1 (over x_3) to split, based on its impact on the network's output. After splitting $x_1 \in \{1,2,3,4,5\}$ into $x_1 \in \{1,2,3\}$ and $x_1 \in \{4,5\}$, we apply forward analysis to each of these two new partitions.

As shown in Fig. 3, while the partition for $x_1 \in \{1,2,3\}$ remains *undecided*, the partition for $x_1 \in \{4,5\}$ is certified as *fair*. This partition has 12 pairs of $x \in X$ and $x' \in X$, where $x_2 \neq x'_2$. Therefore, from the input domain X which has 30 (x, x') pairs, we certify $12/30 = 40\%$ as *fair*. Next, we split the *undecided* partition $x_1 \in \{1,2,3\}$ into $x_1 \in \{1,2\}$ and $x_1 \in \{3\}$ and apply forward analysis to each of these two new partitions. While the first new partition remains *undecided*, the second one is certified as *fair*. Since this partition has six (x, x') pairs, it represents $6/30 = 20\%$ of the input domain.

This iterative refinement process continues until one of the following two termination conditions is satisfied: either there is no more partition to apply forward analysis to, or a predetermined time limit (e.g., 30 minutes) is reached.

III. Preliminaries

In this section, we review the fairness definitions as well as the basics of neural network verification.

A. Fairness Definitions

Let $f : X \to Y$ be a classifier, where X is the input domain and Y is the output range. Each input $x \in X$ is a vector in the D-dimensional attribute space, denoted $x = \langle x_1, \ldots, x_D \rangle$, where $1, \ldots, D$ are vector indices. Each output $y \in Y$ is a class label. Some attributes are legally protected attributes (e.g., gender and race) while others are unprotected attributes. Let $\mathcal{P}$ be the set of vector indices corresponding to protected attributes in x. We say that x_j is a protected attribute if and only if $j \in \mathcal{P}$.

Definition 1 (Individual Fairness for a Given Input): Given a classifier f, an input $x \in X$, and a protected attribute $j \in \mathcal{P}$, we say that f is individually fair for x if and only if $f(x) = f(x')$ for any $x' \in X$ that differs from x only in the protected attribute x_j.

This notion of fairness is local in the sense that it requires the classifier to treat an individual x in a manner that is oblivious to its protected attribute x_j of x.

Definition 2 (Individual Fairness for the Input Domain): Given a classifier f, an input domain X, and a protected attribute $j \in \mathcal{P}$, we say that f is individually fair for the input domain X if and only if, for all $x \in X$, $f(x) = f(x')$ holds for any $x' \in X$ that differs from x only in the protected attribute x_j.

This notion of fairness is global since it requires the classifier to treat all $x \in X$ in a manner that is oblivious to the protected attribute x_j of x. The method of explicit enumeration would be prohibitively expensive since the number of individuals in X may be astronomically large or infinite.

B. Connecting Robustness to Fairness

Perturbation robustness, which is the most frequently checked property by existing DNN verifiers, is closely related to the notion of adversarial examples. The idea is that, if the DNN's classification output were robust, then applying a small perturbation to a given input x should not change the classifier's output for x.

Definition 3 (Robustness): Given a classifier f, an input $x \in X$, and a small constant δ, we say that f is robust against δ-perturbation if and only if $f(x) = f(x')$ holds for all $x' \in X$ such that $||x - x'|| \leq \delta$.

By definition, perturbation robustness is a local property defined for a particular input x, where the set of inputs defined by $||x - x'|| \leq \delta$ is not supposed to be large. While in theory, a robustness verifier may be forced to check individual fairness by setting δ to a large value (e.g., to include the entire input domain X), it almost never works in practice. The reason is because, by definition, such a global robustness property requires that all inputs to have the same classification output returned by the DNN – such a classifier f would be practically useless.

This observation has been confirmed by our experiments using α-β-CROWN [9], a state-of-the-art DNN robustness verifier. Toward this end, we have created a merged network that contains two copies of the same network, with one input for one protected attribute group (e.g., male) and the other input for the other group (e.g., female). While the verifier finds counterexamples in seconds (and thus falsifies fairness of the DNN), it has the same limitation as Fairify: it merely declares the DNN as *unsafe* (unfair, in our context) as soon as it finds a counterexample, but does not provide users with any meaningful, quantitative information. In contrast, our method provides a quantitative framework for certified fairness by reasoning about all individuals in the input domain.

IV. Overview of Our Method

In this section, we present the top-level procedure of our method; detailed algorithms of the subroutines will be presented in subsequent sections.

Let the DNN $y = f(x)$ be implemented as a series of affine transformations followed by nonlinear activations, where each affine transformation step and its subsequent nonlinear activation step constitute a hidden layer. Let l be the total number of hidden layers, then $f = f_l(f_{l-1}(...f_2(f_1(x \cdot W_1) \cdot W_2)... \cdot W_{l-1})$. For each $k \in [1, l]$, W_k denotes the affine transformation and $f_k()$ denotes the nonlinear activation. More specifically, W_1 consists of the edge weights at layer 1 and $x \cdot W_1 = \Sigma_i x_i w_{1,i}$. Furthermore, f_1 is the activation function, e.g., ReLU$(x \cdot W_1) = max(0, x \cdot W_1)$.

A. The Basic Components

Similar to existing *symbolic interval analysis* based DNN verifiers [7], [8], [9], our method consists of three basic components: forward analysis, classification, and refinement.

a) Forward Analysis: The goal of forward analysis is to compute upper and lower bounds of the network's output for all inputs. It starts by assigning a symbolic interval to each input attribute. For example, in Fig. 2, $i_1 = [x_1, x_1]$ is symbolic, where $x_1 \in \{0, 1, 2, 3, 4, 5\}$. Compared to concrete values, symbolic values have the advantages of making the analysis faster and more scalable. They are also sound in that they overapproximate the possible concrete values.

b) Classification: In a binary classifier, e.g., the DNN in Fig. 2 for making hiring decisions, the output is a singular node o whose numerical values needs to be turned to either the positive or the negative class label based on a threshold value (say 0). For example, if $o \in [0.3, 0.4]$, the output label is guaranteed to be positive since $o > 0$ always holds, and if $o \in [-0.7, -0.6]$, the output label is guaranteed to be negative since $o < 0$ always holds. However, if $o \in [-0.7, 0.4]$, the output label remains *undecided* – this is when our method needs to conduct refinement.

c) Refinement: The goal of refinement is to partition the input domain of the DNN to improve the (upper and lower) symbolic bounds computed by forward analysis. Since approximation error may be introduced when linear bounds are pushed through nonlinear activation functions (e.g., unless ReLU is always on or always off), by partitioning the input domain, we hope to increase the chance that activation functions behave similar to their linear approximations for each of the new (and smaller) input partitions, thus reducing the approximation error.

B. The Top-Level Procedure

Algorithm 1 shows the top-level procedure, whose input consists of the network f, the protected attribute x_j, and the input domain X. Together, these three parameters define the fairness certification problem, denoted $\langle f, x_j, X \rangle$.

Within the top-level procedure, we first initialize the input partition P as X and push it into the stack S. Each input partition is associated with a refinement depth. Since P is initially the entire input domain X, its refinement depth is set to 0. Subsequently, the refinement depth increments every time P is bisected to two smaller partitions. In general, the refinement depth of $P \subseteq X$ is the number of times that X is bisected to reach P.

Algorithm 1: Overview Method of FairQuant

Input: neural network f, protected attribute x_j, input domain X
Result: $r_{cer}, r_{fal}, r_{und}$, which are percentages of certified, falsified, and undecided inputs

1 Initial partition $P \leftarrow X$ with refinement depth 0
2 Push P into an empty stack S
// initially, 100% undecided
3 $r_{cer} \leftarrow 0, r_{fal} \leftarrow 0, r_{und} \leftarrow 1$
4 **while** S *is not empty and not-yet-timed-out* **do**
5 Pop P from the stack S
// certify current partition
6 $result \leftarrow$ SymbolicForward(f, x_j, P)
7 **if** *result* = *Undecided* **then**
// split current partition
8 $P_l, P_u \leftarrow$ BackwardRefinement(f, P)
9 Push P_l and P_u into the stack S
// update the percentages
10 $r_{cer}, r_{fal}, r_{und} \leftarrow$ QuantifyFairness$(result, P, X)$
11 **return** $r_{cer}, r_{fal}, r_{und}$

In Lines 4-10 of Algorithm 1, we go through each partition stored in the stack S, until there is no partition left or a time limit is reached. For each partition P, we first apply symbolic forward analysis (Line 6) to check if the DNN f is fair for all individuals in P. There are three possible outcomes: (1) *fair* (certified), meaning that $f(x) = f(x')$ for all $x \in P$ and its counterpart x'; (2) *unfair* (falsified), meaning that $f(x) \neq f(x')$ for all $x \in P$ and its counterpart x'; or (3) *undecided*.

Next, if the result is undecided (Line 7), we apply backward refinement by splitting P into two disjoint new partitions P_l and P_u. By focusing on each of these smaller partitions in a subsequent iteration step, we hope to increase the chance of certifying it as fair (or falsifying it as unfair).

Finally, we quantify fairness (Line 10) by updating the percentages of certified (r_{cer}), falsified (r_{fal}) and undecided (r_{und}) inputs of X. Specifically, if the previously-undecided partition P is now certified as *fair*, we decrease the undecided rate r_{und} by $|P|/|X|$ and increase the certified rate r_{cer} by the same amount. On the other hand, if P is falsified as *unfair*, we decrease r_{und} by $|P|/|X|$ and increase the falsified rate r_{fal} by the same amount.

In the next three sections, we will present our detailed algorithms for forward analysis (Section V), backward refinement (Section VI), and quantification (Section VII).

C. The Correctness

Before presenting the detailed algorithms, we would like to make two claims about the correctness of our method. The first claim is about the *qualitative* result of forward analysis, which may be fair, unfair, or undecided.

Theorem 1: When forward analysis declares an input partition $P \subseteq X$ as *fair*, the result is guaranteed to be sound in that $f(x) = f'(x)$ holds for all $x \in P$ and its counterpart

x', Similarly, when forward analysis declares P as *unfair*, the result is guaranteed to be sound in that $f(x) \neq f'(x)$ holds for all $x \in P$ and its counterpart x'.

The above soundness guarantee is because SYMBOLICFORWARD soundly overapproximates the DNN's actual behavior. That is, the upper bound UB is possibly-bigger than the actual value, and the lower bound LB is possibly-smaller than the actual value. As a result, the symbolic interval $[LB, UB]$ computed by SYMBOLICFORWARD guarantees to include all concrete values. In the next three sections, we shall discuss in more detail how the symbolic interval is used to decide if P is *fair*, *unfair*, or *undecided*.

When an input partition P is *undecided*, it means that some individuals in P may be treated fairly whereas others in P may be treated unfairly. This brings us to the second claim about the *quantitative* result of our method, represented by the rates r_{cer}, r_{fal} and r_{und}.

Theorem 2: The certification rate r_{cer} computed by our method is guaranteed to be a lower bound of the percentage of inputs whose outputs are actually fair. Similarly, the falsified rate r_{fal} is a lower bound of the percentage of inputs whose outputs are actually unfair.

In other words, when our method generates the percentages of *fair* and *unfair* inputs, it guarantees that they are provable lower bounds of certification and falsification, respectively. The reason is because SYMBOLICFORWARD soundly overapproximates the actual value range. When the output intervals indicate that the model is *fair* (*unfair*) for all inputs in P, it is definitely *fair* (*unfair*). Thus, both r_{cer} and r_{fal} are guaranteed to be lower bounds.

Since the sum of the three rates is 1, meaning that $r_{und} = 1 - r_{cer} - r_{fal}$, the undecided rate r_{und} is guaranteed to be an upper bound.

V. SYMBOLIC FORWARD ANALYSIS

Algorithm 2 shows our forward analysis subroutine, which takes the subproblem $\langle f, x_j, P \rangle$ as input and returns the certification result as output.

Algorithm 2: Subroutine SYMBOLICFORWARD()

Input: neural network f, protected attribute x_j, input partition P
Result: certification $result$, which may be fair/unfair/undecided

$I \leftarrow P|_{x_j \in [0,0]}$ and $I' \leftarrow P|_{x_j \in [1,1]}$
$O \leftarrow$ FORWARDPASS(f, I) // for $x \in P$ with $x_j \in [0, 0]$
$O' \leftarrow$ FORWARDPASS(f, I') // for $x' \in P$ with $x_j \in [1, 1]$
if $(O_{lb} > 0 \wedge O'_{lb} > 0) \vee (O_{ub} < 0 \wedge O'_{ub} < 0)$ **then**
  $result \leftarrow$ Fair
else if $(O_{lb} > 0 \wedge O'_{ub} < 0) \vee (O_{ub} < 0 \wedge O'_{lb} > 0)$ **then**
  $result \leftarrow$ Unfair
else
  $result \leftarrow$ Undecided
return $result$

A. The Two Steps

Our forward analysis consists of two steps. First, a standard *symbolic interval* based analysis is invoked twice, for the symbolic inputs I and I', to compute the corresponding symbolic outputs O and O'. Second, O and O' are used to decide if the certification result is *fair*, *unfair*, or *undecided*.

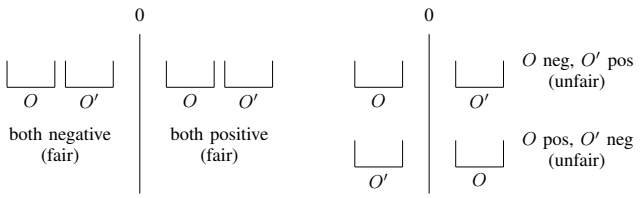

Fig. 4. Sufficient conditions for deciding fairness based on symbolic output intervals O and O', and the threshold 0: there are two fair conditions (left) and two unfair conditions (right).

In the first step, the symbolic input $I = P|_{x_j \in [0,0]}$ is defined as the subset of input partition P where all inputs have the protected attribute x_j set to 0. In contrast, $I' = P|_{x_j \in [1,1]}$ is defined as the subset of P where all inputs have x_j set to 1. The output O is a sound overapproximation of $f(x)$ for $x \in I$, whereas the output O' is a sound overapproximation of $f(x')$ for $x' \in I'$. The subroutine FORWARDPASS used to compute O and O' is similar to any state-of-the-art neural network verifier based on symbolic interval analysis; in our implementation, we used the algorithm of ReluVal [7].

In the second step, the two output intervals, $O = [O_{lb}, O_{ub}]$ and $O' = [O'_{lb}, O'_{ub}]$, are used to compute the certification result. To understand how it works, recall that in the concrete domain, the numerical value of the DNN's output node is compared against a threshold, say 0, to determine if the output label should be positive or negative. In the symbolic interval abstract domain, the upper and lower bounds of the numerical values are used to determine if the model is fair, unfair, or undecided.

Below are the five scenarios:

1) If $O_{lb} > 0$ and $O'_{lb} > 0$, both O and O' have the positive label, meaning that f is fair for P.
2) If $O_{ub} < 0$ and $O'_{ub} < 0$, both O and O' have the negative label, meaning that f is fair for P.
3) If $O_{lb} > 0$ and $O'_{ub} < 0$, O is positive but O' is negative, meaning that f is unfair for P.
4) If $O_{ub} < 0$ and $O'_{lb} > 0$, O is negative but O' is positive, meaning that f is unfair for P.
5) Otherwise, f remains undecided for P.

Fig. 4 illustrates the first four scenarios above. Specifically, the horizontal line segments represent the value intervals of O and O', whose upper/lower bounds may be either > 0 or < 0. The vertical lines represent the threshold value 0.

We can extend out method from two protected attribute (PA) groups (e.g., male and female) to more than two PA groups. For example, if the protected attribute x_j has three values, we will have three symbolic inputs (I, I' and I'') and three corresponding symbolic outputs (O, O' and O''). To decide if f is fair (or unfair) in this multi-PA group setting, we check if (1) individuals in each PA group receive the same output label; and (2) the output labels for the three PA groups are the same.

We can also extend our method from binary classification to multi-valued classification. For example, if there are three possible output labels, we will have O_1, O_2, and O_3 as the symbolic intervals for the three values for one PA group (I) and O'_1, O'_2 and O'_3 for the other PA group (I'). To decide if f is fair (or unfair) for this multi-valued classification, we check (1) which output labels are generated for I and I'; and (2) whether these two output labels (O and O') are the same.

B. The Running Example

For our running example in Fig. 2, consider the initial input partition $P = X$. For ease of understanding, we denote the symbolic expressions for a neuron n as $S_{in}(n)$ after the affine transformation, and as $S(n)$ after the ReLU activation. Furthermore, S will be used for I, and S' will be used for I'.

Let $I = P|_{x_j=0}$ and $I' = P|_{x_j=1}$. After affine transformation in the hidden layer, we have $S_{in}(h_1) = 2x_1 + 1.2x_3$ and $S'_{in}(h_1) = 2x_1 + 1.2x_3 + 0.5$. If we concretize these symbolic expressions, we will have $S_{in}(h_1) = [2, 16]$ and $S'_{in}(h_1) = [2.5, 16.5]$. Based on these concrete intervals, we know that h_1 is always active for both I and I'. Since the activation function is ReLU, we have $S(h_1) = S_{in}(h_1)$ and $S'(h_1) = S'_{in}(h_1)$.

For the hidden neuron h_2, we have $S_{in}(h_2) = -0.2x_1 + 0.4x_3$ and $S'_{in}(h_2) = -0.2x_1 + 0.4x_3 + 0.7$, whose corresponding concrete bounds are $[-1, 1.8]$ and $[-0.3, 2.5]$, respectively. In both cases, since h_2 is nonlinear (neither always-on nor always-off), we must approximate the values using linear expressions to obtain $S(h_2)$ and $S'(h_2)$. While we use the sound overapproximation method of Wang et al. [20], other techniques (e.g., [21], [8]) may also be used.

After overapproximating the ReLU behavior of h_2, we obtain $S(h_2) = [-0.128x_1 + 0.257x_3,\ -0.128x_1 + 0.257x_3 + 0.643]$ and $S'(h_2) = [-0.178x_1 + 0.357x_3 + 0.625,\ -0.178x_1 + 0.357x_3 + 0.893]$.

Finally, we compute $S_{in}(o) = [0.528x_1 - 0.017x_3 - 0.643,\ 0.528x_1 - 0.017x_3]$ and $S'_{in}(o) = [0.578x_1 - 0.117x_3 - 0.793,\ 0.578x_1 - 0.117x_3 - 0.525]$. From these symbolic bounds, we obtain the concrete bounds of $O = [-0.2, 2.64]$ and $O' = [-0.8, 2.368]$. Since these output intervals are not tight enough to determine the output labels for I and I', which are needed to decide if the model is fair or unfair for the partition P, the model remains undecided.

To improve the accuracy, we need to split P into smaller input partitions and then apply symbolic forward analysis to each partition again. How to split P will be addressed by the iterative backward refinement method presented in the next section.

VI. Iterative Backward Refinement

The goal of iterative backward refinement is to split the currently-undecided input partition P into smaller partitions, so that for each of these smaller partitions, symbolic forward analysis will obtain a more accurate result. Algorithm 3 shows the pseudo code, which takes the network f and the partition P as input and returns two smaller partitions P_l and P_u as output. Inside this procedure, Lines 7-14 are related to splitting P, and Lines 2-6 are related to early termination conditions.

Algorithm 3: Subroutine BackwardRefinement()

Input: neural network f, input partition P
Result: smaller partitions P_l and P_u, if any

```
1  Let cex_count be the total number of counterexamples found
2  if (P.depth ≥ max_refinement_depth) then
3      return (null,null)                                  // do not split P
4  else if (P.depth ≥ min_sample_depth) and SampledCEX(P) then
5      cex_count += 1
6      return (null,null)                                  // do not split P
7  else
       // R, R' are gradient masks computed for I, I'
8      g_I ← BackwardPass(f, R)
9      g_I' ← BackwardPass(f, R')
10     g = (g_I + g_I')/2
       // Best input attribute to bisect
11     x_i ← argmax_{x_i} g(x_i) * |ub(x_i) − lb(x_i)|
12     P_l ← P | x_i ∈ [lb, (lb+ub)/2]
13     P_u ← P | x_i ∈ [(lb+ub)/2, ub]
14     return P_l, P_u
```

A. Early Termination Conditions

In Lines 2-6 of Algorithm 3, we check if $P.depth$ exceeds the predefined *max_refinement_depth*. If the answer is yes, we avoid splitting P further. For example, if *max_refinement_depth*$= 20$, it means the current partition P occupies only $|P|/|X| = \frac{1}{2^{20}}$ of the entire input domain X. By increasing the refinement depth, we can decrease the percentage of undecided inputs over X.

If $P.depth$ has not exceeded the maximal refinement depth, we check if $P.depth$ exceeds the predefined *min_sample_depth*, which is set to a value (e.g., 15) smaller than *max_refinement_depth*. When $P.depth$ exceeds this threshold, we start searching for counterexamples in P via random sampling.

Inside the random sampling subroutine SampledCEX(P) shown in Line 4, we sample up to 10 concrete inputs in P and check if x and its counterpart x' satisfy $f(x) \neq f(x')$. If this condition is satisfied, a counterexample is found (but P remains undecided); in this case, we increment *cex_count* and stop splitting P. If no counterexample is found, we continue splitting P into smaller partitions.

Note that in both early termination cases (Lines 3 and 6), the partition P will be marked as *undecided* since we are not able to decide whether the DNN model is fair or unfair to *all individuals* in P.

B. Splitting Input Intervals

In Lines 7-14 of Algorithm 3, we split P into smaller partitions P_l and P_u by first identifying the input attribute x_i that has the largest influence on the output (Lines 8-12) and then bisecting its input interval $x_i \in [lb, ub]$.

Our method for identifying the input attribute x_i is based on maximizing the impact of an input attribute on the network's output. One way to estimate the impact is taking the product of the gradient $g(x_i)$ and the input range $|ub(x_i) - lb(x_i)|$. In

the literature, the product is often called the *smear value* [22], [23]. Unlike existing methods such as Wang et al. [7], however, our computation of the smear value is different because we must consider both inputs I and I', which may have different gradients.

Specifically, during forward analysis, we store the neuron activation information in two gradient mask matrices denoted R and R', where $R[i][j]$ is [1,1] if the j-th neuron at i-th layer is always active, [0,0] if it is always inactive, and [0,1] if it is unknown. The neuron activation information is used later to perform backward refinement for this partition P.

During refinement, we first compute the two gradients g_I and $g_{I'}$ and then take the average. Our goal is to identify the input attribute that has the largest *overall* influence on the network's output.

C. The Running Example

Consider our running example in Fig. 2 again. To compute the smear value, we start with the output layer's edge weights, which are 0.2 for h_1 and -1 for h_2. Since the ReLU associated with h_1 is always-on, $g_I(h_1)$ and $g_{I'}(h_1)$ are set to the interval $[0.2, 0.2]$. However, since the ReLU associated with h_2 is nonlinear, as indicated by the gradient mask matrices R and R', $g_I(h_2)$ and $g_{I'}(h_2)$ are set to the interval $[-1.0, 0]$.

Then, we propagate these gradient intervals backwardly, to get $g_I(i_1) = g_{I'}(i_1) = [(0.2 \times 2) + (0 \times -0.2), (0.2 \times 2) + (-1 \times -0.2)] = [0.4, 0.6]$ and $g_I(i_2) = g_{I'}(i_2) = [-0.6, 0.1]$ and $g_I(i_3) = g_{I'}(i_3) = [-0.16, 0.24]$.

Next, we compute the average g, based on which we compute the smear values. Since x_1 has the smear value of $0.6 \times 4 = 2.4$ and x_3 has the smear value of $0.24 \times 5 = 1.2$, we choose to partition P by bisecting the input interval of x_1. This leads to the smaller partitions shown in Fig. 3.

D. Generalization

While we only consider ReLU networks in this paper, our refinement technique can be extended to non-ReLU activations. Recall that, by definition, ReLU$(z) = 0$ (inactive) if $z < 0$, and ReLU$(z) = 1$ (active) if $z > 0$. Let $\sigma(z)$ be a non-ReLU activation function. To compute the gradient mask matrices R and R', we use thresholds (ϵ_1, ϵ_2) to approximate the on/off behavior: the mask is [0,0] (inactive) if $z < \epsilon_1$ and [1,1] (active) if $z > \epsilon_2$.

Although the approximate on/off behavior of non-ReLU activation function $\sigma(z)$ is not the same as the on/off behavior of ReLU(z), it serves as a practically-useful heuristic to rank the input attributes. Furthermore, this generalization will not affect the soundness of our method, since the gradient masks computed in this manner are only used for picking which input attribute to split first.

VII. Fairness Quantification

We now present our method for updating the percentages of certified and falsified inputs, when the DNN model is found to be fair or unfair for the current input partition P. The pseudo code is shown in Algorithm 4.

Algorithm 4: Subroutine QUANTIFYFAIRNESS()

Input: certification $result$, input partition P, input domain X
Result: Percentages $r_{cer}, r_{fal}, r_{und}$

```
partition_size = ∏_{∀x_i∈P, x_i≠x_j} (UB(x_i) − LB(x_i))
domain_size = ∏_{∀x_i∈X, x_i≠x_j} (UB(x_i) − LB(x_i))
// percentage of inputs in partition P
r_P = (partition_size / domain_size)
if result = Fair then
    r_cer += r_p
    r_und −= r_p
else if result = Unfair then
    r_fal += r_P
    r_und −= r_P
return r_cer, r_fal, r_und
```

There are three cases. First, if the current partition P is found to be fair, meaning that all inputs in P are treated fairly, we compute the percentage of input domain X covered by the partition P, denoted r_P, and then add r_P to r_{cer}, the percentage of certified inputs. Second, if the current partition P is found to be unfair, meaning that all inputs in P are treated unfairly, we add r_P to r_{fal}, the percentage of falsified inputs. In both cases, we also subtract r_p from r_{und}. Otherwise, the current partition P remains undecided and the percentages remain unchanged.

Consider our running example with input partition P defined as $x_1 \in [4, 5] \wedge x_2 \in [0, 1] \wedge x_3 \in [0, 5]$, as shown by the right child of the root node in Fig. 3. This partition has a total of 24 individuals, and its corresponding $I = P|_{x_2=0}$ and $I' = P|_{x_2=1}$ contain 12 individuals each. In contrast, the entire input domain X has 60 individuals, or 30 pairs of x and its counterpart x' (where $x_2 \neq x'_2$).

For this input partition P, $O = [1.49, 2.65]$ and $O' = [1.0, 2.36]$ are the output intervals. Assuming that the decision threshold is 0, the bounds of O and O' imply that the DNN model will generate the positive label for both I and I', meaning that the DNN model is fair for all individuals in P.

Since the input partition size is 12 and the input domain size is 30, the rate $r_P = \frac{12}{30} = 40\%$. After certifying P to be fair, we can add 40% to r_{cer}, the certification rate, and consequently subtract 40% from r_{und}, the undecided rate.

While the above computation assumes that population distribution for each feature is uniform and thus the percentage (e.g., 40%) is computed directly from the partition size (e.g., 12) and the domain size (e.g., 30), the method can be easily extended to consider a non-uniform population distribution. Furthermore, note that the method works regardless of whether the input attributes have integer or real values.

VIII. Experiments

We have implemented FairQuant in a software tool written in C, by leveraging the OpenBlas[2] library for fast matrix multiplication and symbolic representation of the upper and lower bounds. Our forward analysis follows that of Wang et al. [7], [20]. For experimental comparison, we also run Fairify

[2]http://www.openblas.net

TABLE I
STATISTICS OF THE DATASETS AND DNNS USED IN OUR EXPERIMENTS.

Dataset (PA)	# Inputs	DNN	# Layers	# Neurons	Accuracy (%)
Bank (age)	16	BM-1 [10]	2	80	89.20
		BM-2 [10]	2	48	88.76
		BM-3 [10]	1	100	88.22
		BM-4 [10]	3	300	89.55
		BM-5 [10]	2	32	88.90
		BM-6 [10]	2	18	88.94
		BM-7 [10]	2	128	88.70
		BM-8 [10]	5	124	89.20
German (age)	20	GC-1 [10]	1	50	72.67
		GC-2 [10]	1	100	74.67
		GC-3 [10]	1	9	75.33
		GC-4 [10]	2	10	70.67
		GC-5 [10]	5	124	69.33
Adult (gender)	13	AC-1 [10]	2	24	85.24
		AC-2 [10]	1	100	84.70
		AC-3 [10]	1	50	84.52
		AC-4 [10]	2	200	84.86
		AC-5 [10]	2	128	85.19
		AC-6 [10]	2	24	84.77
		AC-7 [10]	5	124	84.85
		AC-8 [10]	2	10	82.15
		AC-9 [10]	4	12	81.22
		AC-10 [10]	4	20	78.56
		AC-11 [10]	4	40	79.25
		AC-12 [10]	9	45	81.46
Compas (race)	6	compas-1	2	24	73.46
		compas-2	5	124	72.82
		compas-3	3	600	72.98
		compas-4	9	90	72.98
		compas-5	10	2000	72.01
		compas-6	4	4000	73.95
		compas-7	10	10000	72.49

which is the only currently-available tool for DNN individual fairness certification. Since Fairify cannot quantify the degree of fairness, we compute the certified/falsified/undecided rates based on its reported statistics.

It is worth noting that Fairify and our method (FairQuant) have a fundamental difference in falsification. Fairify stops and declares an input partition as SAT as soon as it finds a counterexample in that partition; thus, the number of counterexamples that it finds is always the same as the number of SAT partitions it reports. However, SAT partitions are not necessarily *unfair* partitions, since *unfair* partitions require all inputs to be counterexamples, but a SAT partition, excluding the one counterexample, still remains *undecided*.

FairQuant checks if an entire partition is *unfair*. Moreover, when the partition is *undecided*, it can minimize the amount of *undecided* inputs by only sampling for counterexamples after it reaches a deep enough refinement depth. This is made possible through our iterative refinement.

For example, in a DNN model named GC-3, Fairify finds 194 SAT partitions (together with 6 UNSAT and 1 UNKNOWN partitions). However, none of these 194 SAT partitions are *unfair* partitions. Instead, the percentage of falsified inputs is close to being 0% (representing 194 counterexamples out of over 435 trillion individuals in the input domain), the percentage of certified inputs is 2.985% (6 UNSAT partitions out of 201 partitions), and the rest remains undecided. FairQuant, on the other hand, finds 25,963 counterexamples; furthermore, it is able to formally certify 58.44% of the inputs as *fair*.

A. Benchmarks

Table I shows the statistics of the benchmarks, including 32 deep neural networks trained on four popular datasets for fairness research. Among the 32 networks, 25 came from Fairify [10] and the other 7 were trained by ourselves using TensorFlow. All of these networks have a single node in the output layer, to determine the binary classification result. Columns 1-2 show the name of each dataset with its considered protected attribute (PA) and the number of input attributes. Columns 3-6 show the name of each DNN model, its number of hidden layers, number of hidden neurons, and classification accuracy. The accuracy for DNNs trained on *Bank*, *German*, and *Adult* was provided by the Fairify paper. For the models we trained using *Compas*, we have reserved 10% of the data for testing. All the networks coming directly from Fairify on *Bank*, *German*, and *Adult* datasets are small, where the largest has only 200 hidden neurons. Moreover, most of them have only 1 or 2 hidden layers. Thus, we additionally trained much larger networks, with up to 10,000 hidden neurons, using the *Compas* dataset.

Details of the four datasets are given as follows. *Bank* [16] is a dataset for predicting if a bank client will subscribe to its marketing; it consists of 45,000 samples. *German* [17] is a dataset for predicting the credit risk of a person; it consists of 1,000 samples. *Adult* [18] is a dataset for predicting if a person earns more than $50,000; it consists of 32,561 samples. Finally, *Compas* [19] is a dataset for predicting the risk of recidivism; it consists of 6,172 samples.[3]

We evaluate our method using three legally-protected input attributes. For *Bank* and *German*, we use *age*; for *Adult*, we use *gender*; and for *Compas*, we use *race*.[4] These are consistent with Fairify and other prior works in the fairness research.

B. Experimental Setup

We ran all experiments on a computer with 2 CPU, 4GB memory, and Ubuntu 20.04 Linux operating system. We set a time limit of 30 minutes for each DNN model. Our experiments were designed to answer three research questions:

1) Is FairQuant more accurate than the current state-of-the-art in certifying individual fairness of a DNN model?
2) Is FairQuant more scalable than the current state-of-the-art in handling DNN models, especially when the network size increases?
3) Is FairQuant more effective than the current state-of-the-art in providing feedback, e.g., by quantitatively measuring the percentages of certified, falsified, and undecided inputs?

Fairify requires a parameter *MS* (maximum size of an input attribute) based on which it creates a fixed number of input partitions prior to certification. On the DNN models trained for *Bank*, *German*, and *Adult*, we used the default *MS* values

[3]We used the preprocessed *Compas* data provided by [24].

[4]For Bank and German, we use binarized *age* attribute provided by Fairify. For Compas, we binarize *race* attribute into {white, non-white} as done in [25].

TABLE II
RESULTS FOR FAIRNESS CERTIFICATION: FAIRIFY VIS-À-VIS FAIRQUANT

Dataset	DNN	**Fairify** [10]						**FairQuant** (new)					
		Time	Cex	#Cex	Cer%	Fal%	Und%	Time	Cex	#Cex	Cer%	Fal%	Und%
Bank	BM-1	30m	✓	11	10.00	0	90.00	4.82s	✓	2820	94.23	0	5.76
	BM-2	31m	✓	28	16.07	0	83.93	3.23s	✓	2479	93.41	0	6.58
	BM-3	31m	✓	27	19.60	0	81.40	1.21s	✓	1864	95.69	0	4.30
	BM-4	35m	✓	4	3.72	0	96.18	71.12s	✓	5135	87.03	0	12.96
	BM-5	23m	✓	114	77.25	0	22.75	1.03s	✓	1474	96.27	0	3.72
	BM-6	12m	✓	155	69.41	0	30.59	0.44s	✓	1426	96.44	0	3.55
	BM-7	30m	✓	57	9.41	0	90.59	12.26s	✓	7017	83.65	0	16.34
	BM-8	30m	✓	1	0.98	0	99.02	18.99s	✓	3074	90.75	0	9.24
German	GC-1	32m	✓	22	0	0	100	9.73s	✓	31585	32.67	0	67.33
	GC-2	33m	✓	6	0	0	100	31.72s	✓	32655	42.21	0	57.79
	GC-3	8m	✓	194	2.98	0	97.02	6.77s	✓	25963	58.44	0	41.55
	GC-4	4m	✓	2	99.00	0	1.00	0.29s	✓	77	99.65	0	0.34
	GC-5	30m	✗	0	0	0	100	1.24s	✓	9	99.80	0	0.19
Adult	AC-1	32m	✓	3	0.03	0	99.97	3.23s	✓	6151	90.68	0	9.31
	AC-2	31m	✓	9	0.01	0	99.99	30.04s	✓	13008	79.93	0	20.06
	AC-3	32m	✓	20	0	0	100	37.12s	✓	60494	33.29	0	66.70
	AC-4	36m	✗	0	0	0	100	8m	✓	61324	24.79	0	75.20
	AC-5	33m	✗	0	0	0	100	4m	✓	71012	19.12	0	80.87
	AC-6	33m	✓	4	0.01	0	99.99	10.20s	✓	31593	58.82	0	41.17
	AC-7	30m	✗	0	0.01	0	99.99	4m	✓	25588	31.72	0	68.27
	AC-8	30m	✓	39	0.03	0	99.97	11.18s	✓	26179	66.50	0	33.49
	AC-9	30m	✓	126	0.64	0	99.36	3.50s	✓	5470	91.13	0	8.86
	AC-10	32m	✓	8	0.03	0	99.97	5.01s	✓	9033	87.65	0	12.34
	AC-11	30m	✗	0	0	0	100	36.44s	✓	24516	58.01	0	41.98
	AC-12	30m	✗	0	0.02	0	99.98	0.91s	✓	8824	70.82	0	29.17
Compas	compas-1	17m	✓	2	80.00	0.32	19.68	0.01s	✓	17	97.27	2.72	0
	compas-2	31m	✗	0	0	0	100	0.01s	✓	15	97.59	2.40	0
	compas-3	30m	✗	0	0	0	100	0.30s	✓	12	98.07	1.92	0
	compas-4	30m	✗	0	0	0	100	0.01s	✓	14	97.75	2.24	0
	compas-5	T/O	✗	0	0	0	100	5.24s	✓	11	98.23	1.76	0
	compas-6	M/O	✗	0	0	0	100	9.19s	✓	12	98.07	1.92	0
	compas-7	M/O	✗	0	0	0	100	101.25s	✓	15	97.59	2.40	0

(100, 100, and 10) for Fairify to create 510, 201, and 16000 partitions, respectively. On the new DNN models trained for *Compas*, we set *MS* to a small value of 2 to create 20 partitions for Fairify. This was done to maximize Fairify's performance such that it does not "choke" in verifying each input partition.

By default, Fairify uses 100 seconds as "soft timeout" for each input partition and uses 30 minutes as "hard timeout" for the entire DNN. This means that it takes at most 100 seconds to verify a single input partition, and if unsolved, it just moves to the next partition, until the entire 30 minutes runs out.

To run FairQuant, we set the parameters *min_check_depth* to 15 and *max_refinement_depth* to 20 for all DNN models. We also use 30 minutes as "hard timeout", but FairQuant always finished before the limit.

C. Experimental Results

Table II shows the results of our method (FairQuant) in comparison with Fairify[5]. Columns 1-2 show the names of the dataset and the DNN model. Columns 3-5 show the statistics reported by Fairify, including the time taken, whether a counterexample is found (*Cex*) and the number of counterexamples found (*#Cex*). T/O or M/O in Column 3 respectively means that Fairify either spent all 30m or ran out of memory in the network pruning step prior to verifying any input partition. Columns 6-8 show the percentage of certified, falsified and undecided inputs (*Cer%, Fal%, Und%*). Columns 9-14 show the corresponding results from FairQuant.

[5]The order in which Fairify sorts the partitions before running the verification query is random and non-deterministic, so there may be minor difference in the reported counterexamples in the original evaluation and ours.

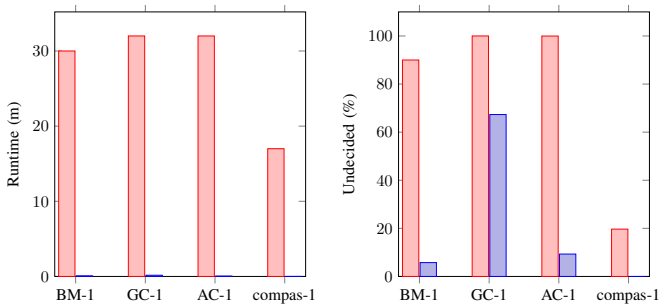

Fig. 5. Comparing the runtime overhead (left) and accuracy (right) of Fairify [10], in red, and FairQuant (new), in blue.

1) Results for RQ 1: To answer the first research question (RQ 1), i.e., whether our method is more accurate than the current state-of-the-art, we need to compare the results shown in Columns 4-5 (for Fairify) with the results shown in Columns 10-11 (for FairQuant). Specifically, Columns 4 and 10 indicate whether the tool is able to find a counterexample (✓) or not (✗) within the time limit.

While our method (FairQuant) found counterexamples for all 32 DNNs, Fairify found counterexamples for only 20 of the 32 models. In addition, it found counterexamples for only one of the 7 newly added models. Moreover, Columns 5 and 11 show that, on models where both tools found counterexamples, the number of counterexamples found by FairQuant is often thousands of times more. For example, the largest number of counterexamples found by Fairify is 194 (for GC-3) but the large number of counterexamples found by FairQuant is 71,012 (for AC-5).

2) Results for RQ 2: To answer the second research question (RQ 2), i.e., whether our method is more scalable than the current state-of-the-art, we need to compare the running time shown in Column 3 (for Fairify) with the running time shown in Column 9 (for FairQuant). While our method (FairQuant) always finished within the time limit of 30 minutes, Fairify timed out on compas-5 and ran out of memory on compas-6 and compas-7. Even on the models where both tools finished, the time taken by Fairify is significantly longer.

To illustrate the scalability advantage of our method, we took a subset of the models for which both FairQuant and Fairify finished, and plot the running time in a bar chart, shown on the left side of Fig. 5. Here, the red bars represent the time taken by Fairify and the blue bars represent the time taken by FairQuant. The results show that FairQuant is many orders of magnitude faster, and can certify DNN models that are well beyond the reach of Fairify.

3) Results for RQ 3: To answer the third research question (RQ 3), i.e., whether our method is more effective in providing feedback to the user, we need to compare the results in Columns 6-8 (for Fairify) with the results in Columns 12-14 (for FairQuant), which show the certified, falsified, and undecided percentages. Since Fairify was not designed to quantitatively measure the degree of fairness, it did poorly in almost all cases. Except for a few DNN models for Bank,

GC-4, and compas-1, its certified percentages are either 0 or close to 0, and its undecided percentages are almost 100%. It means that, for the vast majority of individuals in the input domain, whether they are treated by the DNN model fairly or not remains undecided.

In contrast, the certified percentages reported by our method (FairQuant) are significantly higher. For the models trained using the Compas dataset, in particular, the certified percentages are around or above 90%, and more importantly, the undecided percentages are always 0. It means that FairQuant has partitioned the input domain in such a way that each partition is either certified as being fair, or falsified as being unfair. Even on the subset of DNN models where some inputs remain undecided by FairQuant, the undecided percentages reported by our method are significantly lower than Fairify, as shown on the right side of Fig. 5.

D. Summary

The results show that our method is more accurate and more scalable than the current state-of-the-art techniques for *qualitative* certification. In addition, our method is able to formally *quantify* the degree of fairness, which is a capability that existing methods do not have.

For some DNN models, FairQuant still has a significant percentage of inputs left undecided. This is because we set *min_check_depth=15* and *max_refinement_depth=20* for all benchmarks. Thus, as soon as FairQuant reaches the refinement depth 15 (see the refinement tree shown in Fig. 3) and finds a counterexample in the input partition, it will stop refining further; at that moment, all inputs in the partition are treated *conservatively* as undecided.

In general, a smaller refinement depth allows FairQuant to terminate quickly. During our experiments, FairQuant terminated after 0.29s and 1.24s for GC-4 and GC-5, respectively, compared to the 4 minutes and 30 minutes taken by Fairify and yet returned better results. In fact, for GC-5, Fairify spent 30 minutes but failed to find any counterexample. If we increase the refinement depth, by increasing the two threshold values of FairQuant, its quantification results will get even better.

IX. Related Work

Our method is the first scalable method for certifying and quantifying individual fairness of a deep neural network, and it outperforms the most closely related prior work, Fairify [10]. To the best of our knowledge, no other methods can match the accuracy, scalability, and functionality of our method.

Our method differs from existing techniques for verifying individual fairness properties for neural networks. Libra [31], [32] uses abstract domains to perform verification but is limited in scalability due to the expensive pre-analysis, where a network of 20 hidden nodes takes several hours even with leveraging multiple CPUs. DeepGemini [**?**], which outperforms Libra, is built on top of Marabou [**?**], a SMT-based neural network verification tool, and thus shares the same limitations of Fairify. Furthermore, it only evaluates on networks with up to around 250 hidden neurons. Other works have tackled neural network verification of different definitions of individual fairness. Benussi et al. [26] and Khedr et al. [27] proposed different methods to certify a definition of global individual fairness proposed in [28]. Ruoss et al. [29] verify a type of a *local* individual fairness property that is similar to local robustness, given an input x and a small constant ε for perturbation. This is different from a *global* perspective we have discussed so far in this paper.

Group fairness is yet another type of fairness property, which can be verified using probabilistic techniques [14], [15], [30]. The difference between individual fairness and group fairness is that, while individual fairness requires *similar individuals* to be treated similarly, group fairness requires *similar demographic groups* to be treated similarly.

There are other prior works related to fairness verification of other types of machine learning models [28], [33], [6], but they are not applicable to deep neural networks. Testing techniques can quickly detect fairness violations in machine learning models including neural networks [34], [35], [11], [36], [12], [37], but it does not provide formal guarantee that is important for certain applications. There are also techniques for improving fairness of machine learning models [38], [39], [40], [41], [42], [43], [44], which are orthogonal to our method that focuses on certifying and quantifying fairness of existing DNN models.

At a high level, our method is related to the large number of robustness verifiers for deep neural networks based on interval analysis [7], [20], [8], [45], SMT solving [21], [46], [**?**], [47], and mixed-integer linear programming [48], [**?**]. While these verifiers can decide if a model is robust against adversarial perturbation, they cannot directly certify individual fairness, as explained earlier in Section II. Other neural network verifiers that deal with differential [49], [50], [**?**] or equivalence verification [52] are also different, since they evaluate over two networks instead of one network.

X. Conclusion

We have presented FairQuant, a scalable method for certifying and quantifying individual fairness of a deep neural network over the entire input domain. It relies on sound abstraction during symbolic forward analysis to improve scalability, and iterative refinement based on backward analysis to improve accuracy. In addition to certifying fairness, it is able to quantify the degree of fairness by computing the percentages of inputs whose classification outputs can be certified as fair or falsified as unfair. We have evaluated the method on a large number of DNN models trained using four popular fairness research datasets. The experimental results show that the method significantly outperforms state-of-the-art techniques in terms of both accuracy and scalability, as well as the ability to quantify the degree of fairness.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation (NSF) under grant CCF-2220345. We thank the anonymous reviewers for their constructive feedback.

REFERENCES

[1] A. Castelnovo, R. Crupi, G. D. Gamba, G. Greco, A. Naseer, D. Regoli, and B. S. Miguel Gonzalez, "Befair: Addressing fairness in the banking sector," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 3652–3661, 2020.

[2] J. K. Paulus and D. M. Kent, "Predictably unequal: understanding and addressing concerns that algorithmic clinical prediction may increase health disparities," *NPJ digital medicine*, vol. 3, no. 1, p. 99, 2020.

[3] K. T. Rodolfa, H. Lamba, and R. Ghani, "Empirical observation of negligible fairness–accuracy trade-offs in machine learning for public policy," *Nature Machine Intelligence*, vol. 3, no. 10, pp. 896–904, 2021.

[4] C. Wang, B. Han, B. Patel, and C. Rudin, "In pursuit of interpretable, fair and accurate machine learning for criminal recidivism prediction," *Journal of Quantitative Criminology*, vol. 39, no. 2, pp. 519–581, 2023.

[5] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel, "Fairness through awareness," in *Proceedings of the 3rd innovations in theoretical computer science conference*, pp. 214–226, 2012.

[6] S. Galhotra, Y. Brun, and A. Meliou, "Fairness testing: testing software for discrimination," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, (Paderborn Germany), pp. 498–510, ACM, Aug. 2017.

[7] C. Urban, M. Christakis, V. Wüstholz, and F. Zhang, "Perfectly parallel fairness certification of neural networks," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–30, 2020.

[8] M. J. Kusner, J. Loftus, C. Russell, and R. Silva, "Counterfactual fairness," *Advances in neural information processing systems*, vol. 30, 2017.

[9] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1599–1614, 2018.

[10] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, pp. 1–30, Jan. 2019.

[11] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[12] S. Biswas and H. Rajan, "Fairify: Fairness verification of neural networks," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1546–1558, IEEE, 2023.

[13] R. Angell, B. Johnson, Y. Brun, and A. Meliou, "Themis: automatically testing software for discrimination," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, (Lake Buena Vista FL USA), pp. 871–875, ACM, Oct. 2018.

[14] P. Zhang, J. Wang, J. Sun, G. Dong, X. Wang, X. Wang, J. S. Dong, and T. Dai, "White-box fairness testing through adversarial sampling," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, (New York, NY, USA), p. 949–960, Association for Computing Machinery, 2020.

[15] V. Monjezi, A. Trivedi, G. Tan, and S. Tizpaz-Niari, "Information-theoretic testing and debugging of fairness defects in deep neural networks," in *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23, p. 1571–1582, IEEE Press, 2023.

[16] A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori, "Fairsquare: probabilistic verification of program fairness," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–30, 2017.

[17] O. Bastani, X. Zhang, and A. Solar-Lezama, "Probabilistic verification of fairness properties via concentration," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–27, 2019.

[18] S. Moro, P. Rita, and P. Cortez, "Bank Marketing." UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C5K306.

[19] H. Hofmann, "Statlog (German Credit Data)." UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C5NC77.

[20] B. Becker and R. Kohavi, "Adult." UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[21] J. Angwin, J. Larson, S. Mattu, and L. Kirchner, "Machine bias," 2016.

[22] W. Shiqi, K. Pei, W. Justin, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *32nd Conference on Neural Information Processing Systems (NIPS)*, (Montreal, Canada), 2018.

[23] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pp. 269–286, Springer, 2017.

[24] R. B. Kearfott and M. Novoa III, "Algorithm 681: INTBIS, a portable interval Newton/bisection package," *ACM Transactions on Mathematical Software*, vol. 16, no. 2, pp. 152–157, 1990.

[25] R. B. Kearfott, *Rigorous global search: continuous problems*, vol. 13. Springer Science & Business Media, 2013.

[26] J. A. Adebayo, *FairML: ToolBox for diagnosing bias in predictive modeling*. PhD thesis, Massachusetts Institute of Technology, 2016.

[27] T. Le Quy, A. Roy, V. Iosifidis, W. Zhang, and E. Ntoutsi, "A survey on datasets for fairness-aware machine learning," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 12, no. 3, p. e1452, 2022.

[28] D. Mazzucato and C. Urban, "Reduced products of abstract domains for fairness certification of neural networks," in *Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17–19, 2021, Proceedings 28*, pp. 308–322, Springer, 2021.

[29] X. Xie, F. Zhang, X. Hu, and L. Ma, "Deepgemini: Verifying dependency fairness for deep neural network," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 15251–15259, Jun. 2023.

[30] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, and C. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Computer Aided Verification* (I. Dillig and S. Tasiran, eds.), (Cham), pp. 443–452, Springer International Publishing, 2019.

[31] E. Benussi, A. Patane', M. Wicker, L. Laurenti, and M. Kwiatkowska, "Individual fairness guarantees for neural networks," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22* (L. D. Raedt, ed.), pp. 651–658, International Joint Conferences on Artificial Intelligence Organization, 7 2022. Main Track.

[32] H. Khedr and Y. Shoukry, "Certifair: A framework for certified global fairness of neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 8237–8245, 2023.

[33] P. G. John, D. Vijaykeerthy, and D. Saha, "Verifying individual fairness in machine learning models," in *Conference on Uncertainty in Artificial Intelligence*, pp. 749–758, PMLR, 2020.

[34] A. Ruoss, M. Balunovic, M. Fischer, and M. Vechev, "Learning certified individually fair representations," *Advances in neural information processing systems*, vol. 33, pp. 7584–7596, 2020.

[35] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian, "Certifying and Removing Disparate Impact," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (Sydney NSW Australia), pp. 259–268, ACM, Aug. 2015.

[36] Y. Li, J. Wang, and C. Wang, "Certifying the fairness of KNN in the presence of dataset bias," in *International Conference on Computer Aided Verification. Springer*, 2023.

[37] A. Aggarwal, P. Lohia, S. Nagar, K. Dey, and D. Saha, "Black box fairness testing of machine learning models," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, (New York, NY, USA), p. 625–635, Association for Computing Machinery, 2019.

[38] S. Udeshi, P. Arora, and S. Chattopadhyay, "Automated directed fairness testing," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE '18, (New York, NY, USA), p. 98–108, Association for Computing Machinery, 2018.

[39] H. Zheng, Z. Chen, T. Du, X. Zhang, Y. Cheng, S. Ji, J. Wang, Y. Yu, and J. Chen, "Neuronfair: Interpretable white-box fairness testing through biased neuron identification," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 1519–1531, 2022.

[40] M. Yurochkin, A. Bower, and Y. Sun, "Training individually fair ml models with sensitive subspace robustness," in *International Conference on Learning Representations*, 2020.

[41] J. Wang, Y. Li, and C. Wang, "Synthesizing fair decision trees via iterative constraint solving," in *Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part II*, (Berlin, Heidelberg), p. 364–385, Springer-Verlag, 2022.

[42] X. Gao, J. Zhai, S. Ma, C. Shen, Y. Chen, and Q. Wang, "FairNeuron: Improving deep neural network fairness with adversary games on

selective neurons," in *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, (New York, NY, USA), p. 921–933, Association for Computing Machinery, 2022.

[43] F. Ranzato, C. Urban, and M. Zanella, "Fairness-aware training of decision trees by abstract interpretation," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, (New York, NY, USA), p. 1508–1517, Association for Computing Machinery, 2021.

[44] P. Lahoti, A. Beutel, J. Chen, K. Lee, F. Prost, N. Thain, X. Wang, and E. Chi, "Fairness without demographics through adversarially reweighted learning," *Advances in neural information processing systems*, vol. 33, pp. 728–740, 2020.

[45] T. Hashimoto, M. Srivastava, H. Namkoong, and P. Liang, "Fairness without demographics in repeated loss minimization," in *International Conference on Machine Learning*, pp. 1929–1938, PMLR, 2018.

[46] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," *Advances in neural information processing systems*, vol. 29, 2016.

[47] P. Yang, R. Li, J. Li, C.-C. Huang, J. Wang, J. Sun, B. Xue, and L. Zhang, "Improving neural network verification through spurious region guided refinement," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 389–408, Springer, 2021.

[48] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pp. 97–117, Springer, 2017.

[49] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pp. 3–29, Springer, 2017.

[50] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," *Advances in neural information processing systems*, vol. 29, 2016.

[51] R. Bunel, I. Turkaslan, P. H. S. Torr, M. P. Kumar, J. Lu, and P. Kohli, "Branch and bound for piecewise linear neural network verification," *J. Mach. Learn. Res.*, vol. 21, Jan. 2020.

[52] B. Paulsen, J. Wang, and C. Wang, "Reludiff: Differential verification of deep neural networks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 714–726, 2020.

[53] B. Paulsen, J. Wang, J. Wang, and C. Wang, "Neurodiff: scalable differential verification of neural networks using fine-grained approximation," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 784–796, 2020.

[54] S. Mohammadinejad, B. Paulsen, J. V. Deshmukh, and C. Wang, "DiffRNN: Differential verification of recurrent neural networks," in *Formal Modeling and Analysis of Timed Systems - 19th International Conference, FORMATS 2021, Paris, France, August 24-26, 2021, Proceedings* (C. Dima and M. Shirmohammadi, eds.), vol. 12860 of *Lecture Notes in Computer Science*, pp. 117–134, Springer, 2021.

[55] C. Eleftheriadis, N. Kekatos, P. Katsaros, and S. Tripakis, "On neural network equivalence checking using SMT solvers," in *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 237–257, Springer, 2022.

[56] S. Weisberg, *Applied Linear Regression*. John Wiley & Sons, 3rd ed., 2005.