# Causality-Based Neural Network Repair

Bing Sun
Singapore Management University

Jun Sun
Singapore Management University

Long H. Pham
Singapore Management University

Jie Shi
Huawei Singapore

## ABSTRACT

Neural networks have had discernible achievements in a wide range of applications. The wide-spread adoption also raises the concern of their dependability and reliability. Similar to traditional decision-making programs, neural networks can have defects that need to be repaired. The defects may cause unsafe behaviors, raise security concerns or unjust societal impacts. In this work, we address the problem of repairing a neural network for desirable properties such as fairness and the absence of backdoor. The goal is to construct a neural network that satisfies the property by (minimally) adjusting the given neural network's parameters (i.e., weights). Specifically, we propose CARE (**CA**usality-based **RE**pair), a causality-based neural network repair technique that 1) performs causality-based fault localization to identify the 'guilty' neurons and 2) optimizes the parameters of the identified neurons to reduce the misbehavior. We have empirically evaluated CARE on various tasks such as backdoor removal, neural network repair for fairness and safety properties. Our experiment results show that CARE is able to repair all neural networks efficiently and effectively. For fairness repair tasks, CARE successfully improves fairness by 61.91% on average. For backdoor removal tasks, CARE reduces the attack success rate from over 98% to less than 1%. For safety property repair tasks, CARE reduces the property violation rate to less than 1%. Results also show that thanks to the causality-based fault localization, CARE's repair focuses on the misbehavior and preserves the accuracy of the neural networks.

**ACM Reference Format:**
Bing Sun, Jun Sun, Long H. Pham, and Jie Shi. 2022. Causality-Based Neural Network Repair. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510080

## 1 INTRODUCTION

Neural networks have had discernible achievements in a wide range of applications, ranging from medical diagnosis [66], facial recognition [53], fraud detection [19] and self-driving [6]. While neural networks are demonstrating excellent performance, there has been a growing concern on whether they are reliable and dependable. Similar to traditional decision-making programs, neural networks inevitably have defects and need to be repaired at times. Neural networks are usually inherently black-boxes and do not provide explanations on how and why decisions are made in certain ways. As a result, these defects are more "hidden" compared to traditional decision-making programs [5, 17, 23]. It is thus crucial to develop systematic ways to make sure defects in a neural network are repaired and desirable properties are satisfied.

Similar to the activity known as debugging for traditional software programs, there is often a need to modify a neural network to fix certain aspects of its behavior (whilst maintaining other functionalities). Existing efforts to repair the unexpected behavior of neural networks often focus on retraining with additional data [43, 44]. Although retraining is natural and often effective, retraining a neural network model could be difficult and expensive for real-world applications [23]. More importantly, unlike debugging traditional software program (where we can be reasonably certain that the bug is eliminated after the 'fix'), there is no guarantee that the retrained model eliminates the unwanted behavior. Therefore, techniques for modifying an existing model without retraining will be preferable in certain scenarios. That is, we sometimes would like to apply a small modification on an existing neural network to remove unexpected behaviors whilst retaining most of its well-trained behaviors.

Neural networks may misbehave in different ways. Given a model trained primarily for accuracy, it could be discriminative, i.e., the prediction is more favourable to certain groups thus violating fairness property. In this situation, small adjustment can be applied to the trained network without retraining, i.e., to improve its fairness whilst maintaining the model's accuracy. In another scenario, malicious hidden functionalities (backdoor) could be easily embedded into a neural network if the training data is poisoned [42, 67]. Such a backdoor produces unexpected behavior if a specific trigger is added to an input, whilst presenting normal behavior on clean inputs. In this scenario, one would like to repair the neural network by removing the backdoor while maintaining the correct behavior on clean inputs. A further scenario is that a trained model could violate safety-critical properties on certain inputs, e.g., the output is not within the expected range [33, 34]. In this case, we would like to repair the network by making adjustments to its parameters so that the repaired model satisfies the specified property.

For both traditional software programs and neural networks, debugging and repairing are essentially reasoning over causality. For traditional software programs, the notion of causality is natural and well defined [9, 28, 32], e.g., based on (control and data) dependency. However, the same is not true for neural networks, i.e., a wrong decision could easily be the collective result of all neurons in the network and yet considering that all of them are 'responsible' and should be modified is unlikely to be helpful. Although causality analysis has been applied to interpret machine learning models [8] and verification [71], existing causality analysis typically focuses


*ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA*

ACM ISBN 978-1-4503-9221-1/22/05…$15.00
https://doi.org/10.1145/3510003.3510080

on the causal relation from the input to the model prediction, and not the hidden neurons. Hence it is still not clear how to apply existing definitions of causality for repairing neural networks. In other words, how to repair a neural network to remedy defects based on causality is still largely an open problem.

In this work, we introduce CARE (**CA**usality-based **RE**pair), a general automatic repair algorithm for neural networks. Instead of retraining, CARE applies minor modifications to a given model's parameters (i.e., weights) to repair known defects whilst maintaining the model's accuracy. The defects are defined based on desirable properties that the given neural network fails to satisfy. CARE is a search-based automated program repair technique that first performs fault localization based on causal attribution of each neuron, i.e., it locates those neurons that have the most contribution to the model's defects. Secondly, CARE performs automatic repair by adjusting model parameters related to the identified neurons until the resultant model satisfies the specified property (whilst maintaining accuracy).

We summarize our contributinos as follows.

- We propose and implement CARE as a general automatic repair algorithm, that applies causality-based fault localization and PSO optimization to all layers of neural networks.
- We demonstrate the effectiveness of CARE in the context of three different tasks: 1) fairness improvement, 2) backdoor removal and 3) safety property violation repair.
- We empirically evaluate CARE on multiple neural networks including feed-forward neural networks (FFNNs) and convolutional neural networks (CNNs), trained on multiple benchmark datasets. The results indicate that CARE improves the models' performance over the specified properties significantly and CARE outperforms existing approaches proposed for neural network repair.

The remainder of this paper is organized as follows. In Section 2, we review relevant background and define our problem. In Section 3, we present each step of our approach in detail. In Section 4, we evaluate our approach through multiple experiments to answer multiple research questions. We review related work in Section 5 and conclude in Section 6.

## 2 PRELIMINARY

In this section, we review relevant background and define our research problem.

### 2.1 Neural Network Properties

There are many desirable properties we would like a trained neural network to satisfy besides meeting its accuracy requirement. In this work, we assume $\phi$ to be one of the following three kinds of properties and show that CARE can handle all of them.

*Fairness*: Fairness is a desirable property that potentially has significant societal impact [47]. Due to the fact that machine learning models are data-driven and the training data could be biased (or imbalanced), models trained based on such data could be discriminative [4, 63]. In this work, we define independence-based fairness following [61] as follows.

Definition 2.1 (Independence-based Fairness). *Let $N$ be a neural network and $\xi$ be a positive real-value constant. We write $Y$ as the prediction of $N$ on a set of input features $X$ and $L$ as the prediction set. We further write $F \subseteq X$ as a feature encoding some protected characteristics such as gender, age and race. $N$ satisfies independence-based fairness, with respect to $\xi$, if and only if, $\forall l \in L \, \forall f_i, f_j \in F$ such that $i \neq j$,*

$$| P(Y = l \mid F = f_i) - P(Y = l \mid F = f_j) | \leq \xi \tag{1}$$

Intuitively, Definition 2.1 states that, $N$ is fair as long as the probability difference of a favourable prediction for instances with different values of protected feature is within the threshold $\xi$.

*Absence of backdoor*: With the wide adoption of neural networks in critical decision-making systems, sharing and adopting trained models become very popular. On the other hand, this gives attackers new opportunities. Backdoor attack is one of the neural network attacks that often cause significant threats to the system. Backdoor is a hidden pattern trained into a neural network, which produces unexpected behavior if and only if a specific trigger is added to an input [67]. In classification tasks, the backdoor misclassifies an arbitrary inputs to the same target label, when the associated trigger is applied to the input. Hence another desirable property of a neural network would be 'backdoor free', where the backdoor attack success rate is kept below a certain threshold.

Definition 2.2 (Backdoor Attack Success Rate). *Let $N$ be a backdoored neural network, $t$ be the target label, $X$ be a set of adversarial inputs with the trigger and $N(x)$ be the prediction on $x$. We say attack success rate (SR) is:*

$$SR(t) = P(N(x) = t | x \in X) \tag{2}$$

Intuitively, attack success rate is the percentage of adversarial inputs classified into the target label. Next, we define backdoor-free neural network.

Definition 2.3 (Backdoor-free Neural Network). *Let $N$ be a backdoored neural network and $\xi$ be a positive real-value constant. $N$ satisfies backdoor-free property with respect to $\xi$ and $t$, if $SR(t) < \xi$.*

*Safety*: For neural networks applied on safety-critical systems such as the Airborne Collision Avoidance [33] system, all safety properties must be satisfied strictly. However, due to reasons like limited training data and insufficient training, those critical properties could be violated. Hence, another desirable behavior of a neural network is to satisfy all safety properties or the violation rate is kept below a certain threshold.

Definition 2.4 (Safety Property Violation Rate). *Let $N$ be a neural network, $X$ be a set of inputs and $N(x)$ be the prediction on $x$. Let $\rho$ be the critical safety property that $N$ is expected to satisfy. We say the property violation rate (VR) is:*

$$VR(\rho) = P(N(x) \not\models \rho \mid x \in X) \tag{3}$$

Intuitively, violation success rate is the percentage of inputs that violate the property $\rho$. Next, we define safety property violation-free neural network.

Definition 2.5 (Safety Property Violation-free Neural Network). *Let $N$ be a neural network and $\xi$ be a positive real-value*

*constant. N is safety property violation-free with respect to $\xi$ and $\rho$, if $VR(\rho) < \xi$.*

Example 2.1. *We train a feed-forward neural network N on Census Income dataset (refer to details of the dataset in Section 4.1) to predict whether an individual's income exceeds \$50K per year. The neural network is configured to have 5 hidden layers. We use this network as the running example in this paper. In this example, we focus on a fairness property $\phi$ w.r.t. protected feature* gender*, i.e., the model is considered unfair if the probability difference of a favourable prediction for females and males is greater than* 1%*. Note that this is for illustration purpose only and such a threshold is probably too strict for real-world applications. Given this neural network, the problem is to find a repaired network M with minimal adjustment to N's weight parameters so that M satisfies the fairness property whilst maintaining its accuracy.*

## 2.2 Our Problem

We are now ready to define our problem.

Definition 2.6 (Neural Network Repair Problem). *Let N be a neural network, S be the input space, and $\phi$ be one of the above-mentioned desirable properties. The neural network repair problem is to construct a neural network M such that M satisfies $\phi$ in the input space S and the semantic distance between N and M is minimized.*

We define semantic distance between two neural networks $N$ and $M$ as follows.

Definition 2.7 (Semantic Distance). *Let $N(x)$ be the prediction of neural network N on input x where $x \in S$, the semantic distance between N and M is defined as:*

$$P(N(x) \neq M(x)) \tag{4}$$

Intuitively, the semantic distance between two neural networks is the probability that the predictions of the two models are different on the same input. In this work, we use the model accuracy difference of $N$ and $M$ as a measure of their semantic distance.

# 3 OUR APPROACH

In this section, we present the details of our approach. An overview of our framework is as shown in Figure 1 and Algorithm 1 shows the details of our approach. The first step is to verify whether the given neural network $N$ satisfy property $\phi$. If $\phi$ is satisfied, CARE terminates immediately. Otherwise, CARE proceeds to the next step. It performs causality-based fault localization on all hidden neurons of $N$ and identifies those neurons that have the most contribution to $N$'s unwanted behavior. The third step is to optimize these neurons' weights and generate a repaired network $M$ such that $M$ satisfies the property $\phi$ and is semantically close to $N$.

## 3.1 Property Verification

In this step, we verify neural network $N$ against given property $\phi$. If $N$ satisfies $\phi$, CARE returns $N$ unchanged and terminates. Otherwise, CARE proceeds to repair the neural network. Property verification is not the focus of this work and we adopt existing neural network verification techniques. Recently there have been multiple tools and approaches for solving the above-mentioned neural network verification problems. We omit the details on how different properties are verified and refer the readers to [5, 34, 35, 61]. In our implementation, CARE's fairness verification algorithm is based on [61], backdoor success rate verification is based on the method proposed in [67] and safety property verification is based on the approach proposed in [59].

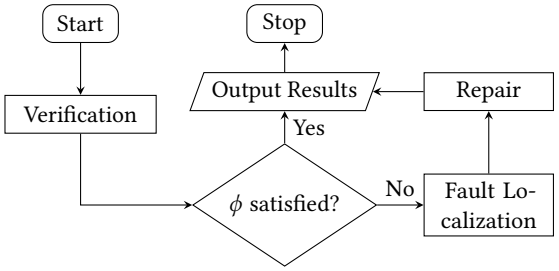


**Figure 1: An overview of our framework**

**Algorithm 1:** $CARE(N, S, \phi)$

```
Verify property φ on N;
Perform accuracy check on N;
if φ is verified then
    return "Verified";
for all hidden neuron x in N do
    Calculate ACE^y_{do(x)};
Sort ACE for all x;
Candidate neurons ← top 10%;
do
    Candidate M ← PSO searching;
    Verify property φ on Candidate M;
    if φ is verified then
        break;
    if Failed to find a better location in last 10 consecutive
     search then
        break;
while #iteration ≤ 100
M = Candidate M;
Perform accuracy check on M;
if Acc(N) − Acc(M) > threshold then
    return "Significant accuracy drop!";
Verify property φ on M;
if φ is verified then
    return M;
else
    return "Not able to repair the network!";
```

Example 3.1. *In our running example, CARE verifies N against the fairness property $\phi$. The resultant fairness score (i.e., the probability difference of a favourable prediction for females and males) is* 1.30%*. Therefore, N fails to satisfy the fairness property (which requires the fairness score to be within* 1%*).*

## 3.2 Causality Analysis

In this step we perform causality-based fault localization. The goal is to identify a relatively small set of neurons to repair. Note that in this work, we restrict ourselves to repair by adjusting the parameters of existing neurons only, i.e., without introducing or pruning neurons. The number of parameters in a neural network is often huge and even relatively small neural network may consist of thousands of weight and bias parameters. Therefore, attempting to optimize all neural weights would be costly and hard to scale. CARE thus adopts a fault localization method to identify neurons that are likely to be responsible for the undesirable behaviors. This method is based on causality analysis carried out over all hidden neurons of $N$.

An alternative method for fault localization is gradient-based method, which is to measure how much perturbing a particular neuron would affect the output [57]. CARE is designed to capture the causal influence of a hidden neuron on the performance on the given property. In contrast, gradient-based method draws conclusion based on statistical correlations, which is not ideal for model repair (since our aim is to identify neurons that *cause* the defect). Next, we describe our causality-based fault localization in detail.

In recent years, causality has gained increasing attention in interpreting machine learning models [8, 27]. Multiple approaches have been designed to explain the importance of the components in a machine learning model when making a decision, based on causal attributions. Compared with traditional methods, causal approaches identify causes and effects of a model's components and thus facilitates reasoning over its decisions.

In the following, we review some concepts which are necessary for the causality analysis in this work.

Definition 3.1 (Structural Causal Models (SCM) [48]). *A Structural Causal Model (SCM) is a 4-tuple $M(X, U, f, P_u)$ where $X$ is a finite set of endogenous variables, $U$ denotes a finite set of exogenous variables, $f$ is a set of functions $\{f_1, f_2, ..., f_n\}$ where each function represents a causal mechanism such that $\forall x_i \in X, x_i = f_i(Pa(x_i), u_i)$ where $Pa(x_i)$ is a subset of $X \setminus \{x_i\}$, $u_i \in U$ and $P_u$ is a probability distribution over $U$.*

SCM serves as a framework for representing what we know about the world, articulating what we want to know and connecting the two together in a solid semantics [49]. It plays an important role in causality analysis and is commonly applied in many studies [8, 45, 46, 73].

Figure 2 shows an example causal graph to study the efficiency of a medication on a disease, where the nodes represent variables and the edges represent cause-effect relations intuitively. In this graph, age is considered as an exogenous variable (i.e., confounder), patient's heart rate, cholesterol level and whether the medication is applied or not are endogenous variables (i.e., whether the medication is applied or not is often considered as the treatment). The outcome is the recovery rate of a patient. As illustrated in the graph, age affects the patient's health conditions such as heart rate and level of cholesterol. Furthermore, the need or feasibility of applying this medicine on patients is affected by age, i.e., young people may not necessarily take the medicine and patients above 70 years old are too risky to take the medicine. Patient's health condition and the application of medication affect the recovery rate. Furthermore,

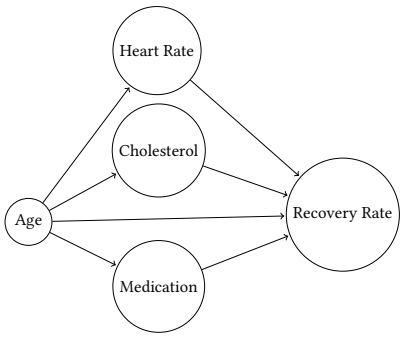


**Figure 2: An Example Causal Graph**

age can affect the recovery rate directly since younger patient often recover faster than the elderly.

In this work, we model neural networks as SCMs to analyze the causal relationship between hidden neurons and model's predictions. To debug and repair a neural network, we would like to measure the 'contribution' of each neuron to the network misbehavior, which is referred as the causal effect.

Definition 3.2 (Average Causal Effect). *The Average Causal Effect (ACE) of a binary random variable $x$ (e.g., treatment), on another random variable $y$ (e.g., outcome) is defined as follows.*

$$ACE = \mathbb{E}[y|do(x = 1)] - \mathbb{E}[y|do(x = 0)] \tag{5}$$

*where $do(\cdot)$ operator denotes the corresponding interventional distribution defined by SCM.*

Intuitively, ACE is used to measure the causal effect of $x$ on $y$ by performing intervention on $x$. There are many other causal effect metrics in the literature such as

- average treatment effect (ATE), i.e., $ATE = \mathbb{E}[y(1) - y(0)]$, where $y(1)$ represents the potential outcome if a treatment is applied exogenously and $y(0)$ represents the corresponding potential outcome without treatment. ATE measures the effect of the treatment at the whole population level [29],
- conditional average treatment effect, i.e., $CATE = \mathbb{E}[y(1) - y(0)|x_0]$ [1], which is the average treatment effect conditioned on a particular subgroup of $X$, i.e., $X = x_0$,
- and effect of treatment on the treated, i.e., $ETT = \mathbb{E}[y_{x_1}|x_0] - E[y|x_0]$ which measures the probability of $Y$ would be $y$ had $X$ been $x_1$ counterfactually, given that in the actual world $X = x_0$ [71].

In this work, we focus on the ACE metric since we are interested in measuring the causal effect of the hidden neurons on certain output value.

## 3.3 Fault Localization

In the following, we present details on how causality analysis is used for fault localization in neural networks. Firstly, the neural network $N$ is modeled as an SCM. As proposed in [38], neural networks can be interpreted as SCMs systematically. In particular, feed-forward neural networks (FFNNs) and convolutional neural networks(CNNs) can be represented as directed acyclic graphs with

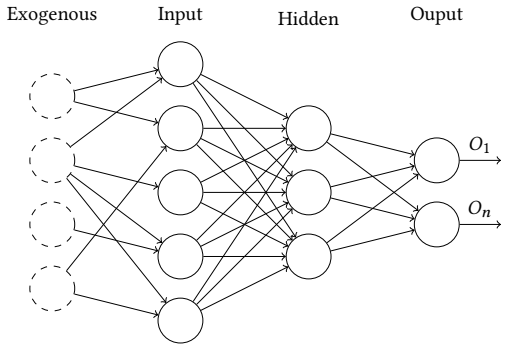


**Figure 3: FFNN as an SCM**

edges from an earlier (i.e., closer to the input layer) layer to the next layer until the output layer. The following is a proposition.

Proposition 3.1. *An n-layer FFNN or CNN $N(x_1, x_2, ..., x_n)$ where $x_i$ represents the set of neurons at layer i, can be interpreted by SCM $M([x_1, x_2, ..., x_n], U, [f_1, f_2, ..., f_n], P_U)$, where $x_1$ represents neurons at input layer and $x_n$ represents neurons at output layer. Corresponding to every $x_i$, $f_i$ represents the set of causal functions for neurons at layer i. $U$ represents a set of exogenous random variables that act as causal factors for input neurons $x_1$ and $P_u$ is a probability distribution over $U$.*

Proof. In the scenario of FFNN, the proof of Proposition 3.1 follows that provided in [8]. In the scenario of CNN, similar to FFNN, neurons at each layer can be written as functions of neurons at its previous layer, i.e., $\forall i: \forall x_{i_j} \in x_i: x_{i_j} = f_{i_j}(x_{i-1})$, where $x_{i_j}$ represents the $j^{th}$ neuron at layer $i$. Neurons at input layer ($x_1$) can be assumed to be functions of independent noise variables $U$ such that $\forall x_{1_j} \in x_1$ *and* $u_j \in U: x_{1_j} = f_{1_j}(u_j)$. Thus a CNN can be equivalently expressed by a SCM $M([x_1, x_2, ..., x_n], U, [f_1, f_2, ..., f_n], P_U)$. □

Figure 3 depicts the SCM of a 3-layer FFNN. The dotted circles represent exogenous random variables which act as causal factors for the input neurons. In this work, we assume that neurons at the input layer are not causally related to each other but can be jointly caused by a latent confounder.

Next we define the attribution problem, i.e., what is the causal influence of a particular hidden neuron on model defect. Recall that in Definition 3.2, we define the ACE of a binary variable $x$ on output variable $y$. However, we cannot apply the definition directly for two reasons. First, the domain of neural networks' neurons is mostly continuous, not binary-valued. Second, we are interested in finding causal effects on the model defect rather than certain output variable. Hence we propose the following definition.

Definition 3.3 (Causal Attribution). *We denote the measure of the undesirable behavior of given neural network N as y. The Causal Attribution of a hidden neuron $x$ to $N$'s defect $y$ is:*

$$ACE^{y}_{do(x=\beta)} = \mathbb{E}[y|do(x = \beta)] \tag{6}$$

Next, we calculate the interventional expectation of $y$ given intervention $do(x = \beta)$ and we thus have the following.

$$\mathbb{E}[y|do(x = \beta)] = \int_y yp(y|do(x = \beta))dy \tag{7}$$

Intuitively, causal attribution measures the effect of neuron $x$ being value $\beta$ on $y$. We evaluate Equation 7 by sampling inputs according to their distribution whilst keeping the hidden neuron $x = \beta$, and computing the average model undesirable behavior $y$.

In this work, we calculate $y$ according to the desirable property $\phi$. Let $N_t(x_{ip})$ be the prediction value of class $t$ on input $x_{ip}$ we have:

- For fairness repair, we measure model unfairness $y$ by taking the difference of the prediction value on favourable class $t$ w.r.t. samples that only differ by the sensitive feature. Let $x_{ip}$ and $x'_{ip}$ be a pair of discriminatory instances [72] (that only differ by the sensitive feature). We have $y_{fair} = \mid N_t(x_{ip}) - N_t(x'_{ip}) \mid$ and we calculate $ACE^{y_{fair}}_{do(x=\beta)}$ as the causal attribution of neuron $x$ w.r.t. fairness property.
- For backdoor removal, we measure $y$ by calculating the prediction value on target class $t$, i.e, $y_{bd} = N_t(x_{ip})$. Thus we aim to calculate $ACE^{y_{bd}}_{do(x=\beta)}$ for all hidden neurons. Note $x_{ip}$ can be clean inputs or adversarial inputs.
- For safety property violation repair, we measure $y$ by calculating the prediction value on desirable class $t$, i.e., $y_{safe} = N_t(x_{ip})$. We calculate $ACE^{y_{safe}}_{do(x=\beta)}$ as the causal attribution for hidden neuron $x$.

For other properties, CARE is able to calculate the causal attribution of each neuron on $y$ as long as the corresponding $y$ is specified.

Thus by calculating the causal attribution of each hidden neuron on $y$, we are able to identify neurons that are most likely the cause to the unexpected behavior. Afterwards, the identified neurons are used as candidates for model repair in our next step. The time spent on this step depends on the total number of neurons $n$ to analyze in the given neural network. The time complexity for causality-based fault localization is thus $\mathbf{O}(n)$.

Example 3.2. *In our running example, CARE conducts causality-based fault localization on all hidden neurons of N to identify neurons that are the most "responsible" to N's prediction bias. CARE generates 12.4k samples to analyze all neurons and the total time taken is 120s. It is observed that a few neurons (such as the $11^{th}$ neuron at $3^{rd}$ hidden layer and the $3^{rd}$ neuron at $5^{th}$ hidden layer) have a contribution which is "outstanding". This is good news to us as it implies that it is possible to improve the fairness by modifying the weights of only a few neurons. CARE then selects the top 10% of total number of neurons (i.e., 13 in total for this network) to be used as repair candidates.*

## 3.4 Network Repair

Next, we present how CARE performs neural network repair. Similar to traditional software programs, neural networks can go wrong and incorrect behaviors should be fixed. While existing efforts mainly focus on how to retrain the neural network to repair the unexpected behaviors, there are a few recent works [3, 23, 65] that address this problem by modifying the parameters of the network.

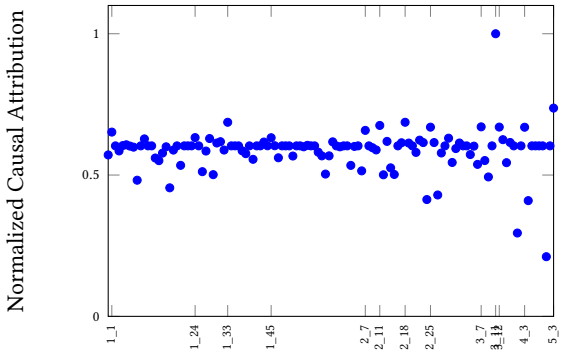


**Figure 4: Causal Attribution of Neurons in the Example**

In the following, we first briefly explain why these approaches do not apply in our setting. Goldberger *et al.* propose a verification-based method to adjust the neural weights of the output layer only. Their approach has limited effectiveness in repairing neural networks as demonstrated in [10]. Their heuristics (i.e., focusing on neurons in the output layer) is not justified from our point of view as the neurons causing the problematic behaviors may not be in the output layer. In [3], an approach is proposed to repair a model through inductive synthesis. Their approach has scalability issues as a large model size leads to a large search space. While NNRepair [65] provides a constraint-based approach to fix the logic of a neural network at an intermediate layer or output layer, its fault localization and repair applies to a single layer only but the actual fault could be across multiple layers. Thus its repair performance may not be ideal and we provide a comparison in Section 4.2.

In our approach, we propose to address the repair problem through optimization, i.e., based on the Particle Swarm Optimisation (PSO) algorithm. Besides PSO, there are many other optimization algorithms. Genetic optimization algorithms (genetic algorithm for example) usually do not handle complexity in an efficient way [54] since the large number of elements undergoing mutation causes a considerable increase in the search space. In comparison, PSO requires smaller number of parameters and thus has lower number of iterations. Another type of optimization is stochastic optimization algorithm. Markov Chain Monte Carlo (MCMC) as an representative uses a sampling technique for global optimization. However, MCMC often encounter either slow convergence or biased sampling [18]. Hence, in this work we select intelligence based algorithm PSO.

The idea is to search for small adjustments in weight parameters of the neurons identified in the previous step such that the specified property is satisfied. PSO simulates intelligent collective behavior of animals such as schools of fish and flocks of birds. It is known to be particularly effective for optimization in continuous domains [52]. In PSO, multiple particles are placed in the search space. At each time step, each particle updates its location $\overrightarrow{x_i}$ and velocity $\overrightarrow{v_i}$ according to an objective function. That is, the velocity is updated based on the current velocity $\overrightarrow{v_i}$, the previous best location found locally $\overrightarrow{p_i}$ and the previous best location found globally $\overrightarrow{p_g}$. Its location is updated based on the current location and velocity. We write $R(0, c)$ to denote a random number uniformly sampled from the range of $[0, c]$. Formally, the PSO update equation is as follows [55].

$$\overrightarrow{v_i} \leftarrow \omega\overrightarrow{v_i} + R(0, c_1)(\overrightarrow{p_i} - \overrightarrow{x_i}) + R(0, c_2)(\overrightarrow{p_g} - \overrightarrow{x_i}) \tag{8}$$

$$\overrightarrow{x_i} \leftarrow \overrightarrow{x_i} + \overrightarrow{v_i} \tag{9}$$

where $\omega$, $c_1$ and $c_2$ represent inertia weight, cognitive parameter and social parameter respectively. In PSO, the fitness function is used to determine the best location.

In CARE, the weights of the identified neurons are the subject for optimization and thus are represented by the location of the particles in PSO. The initial location of each particle is set to their original weight and the initial velocity is set to zero.

As defined in Section 2, our problem is to repair a given neural network $N$ against property $\phi$ while minimizing the accuracy drop. Therefore, two essential components need to be considered in the optimization process: 1) model performance on property $\phi$ and 2) model accuracy. In CARE, we measure the model performance based on the property $\phi$ specified. Formally, the fitness function of PSO is defined as follows.

$$fitness = (1 - \alpha)\ UB + \alpha(1 - accuracy) \tag{10}$$

where UB (undesirable behavior) is a quantitative measure on the degree of violating property $\phi$; constant parameter $\alpha \in (0, 1)$ determines the relative importance of the accuracy. For fairness improvement task, we randomly sample a large set of instances and $UB$ is measured by the percentage of individual discriminatory instances within the set, i.e., let $N(x)$ be the prediction of $x$ and $(x, x')$ be a pair of discriminatory instances in the sample set, $UB = P(N(x) \neq N(x'))$; for backdoor removal task, we add backdoor trigger (if it is known) to all samples in testing set and measure $UB = SR(t)$ following Definition 2.2; for safety task, we randomly sample a large set of instances and measure $UB = VR(\rho)$ following Definition 2.4. Intuitively, our optimization target is to make the repaired model satisfy given property $\phi$ while maintaining the accuracy. Note that, for other properties, CARE is able to conduct repair as long as a performance indicator is specified based on the given property properly.

Example 3.3. *In our running example, CARE applies PSO to generate a repair network M that will improve the fairness performance of N. The weights of the identified 13 neurons in the previous step are the subjects for the optimization. We set α to* 0.8 *so that PSO can optimize for fairness without sacrificing model accuracy too much. PSO terminates at* $17^{th}$ *iteration where no better location is found in the last 10 consecutive iterations. By adjusting neural weights accordingly, M is generated. CARE performs fairness and accuracy check on M and the maximum probability difference is* 0.007 *with accuracy of* 0.86 *(original accuracy is* 0.88*). Thus, CARE successfully repairs N with a fairness improvement of* 46.1% *and the model accuracy is mildly affected.*

## 4 IMPLEMENTATION AND EVALUATION

In the following, we conduct a set of experiments to evaluate CARE. We demonstrate the technique in the context of three different tasks: 1) neural network fairness improvement, 2) neural network backdoor removal and 3) neural network safety property violation

repair. All experiments are conducted on a machine with 1 Dual-Core Intel Core i5 2.9GHz CPU and 8GB system memory. To reduce the effect of randomness, all experimental results are the average of five runs if randomness is involved.

## 4.1 Experiment Setup

In the following, CARE is evaluated over multiple datasets by answering mutpile research questions (RQs). The details of the datasets are summarized as follows:

For fairness repair, CARE is evaluated over three datasets that are commonly used in machine learning model fairness testing [2, 21, 64, 72],

- *Census Income [12]:* This dataset consists of 32,561 training instances containing 13 features and is used to predict whether an individual income exceeds $50K per year. Among all attributes, gender, age and race are three protected characteristics. The labels are if an individual income exceeds $50K per year or not.
- *German Credit [13]:* This dataset consists of 1000 instances with 20 features and is used to assess an individual's credit. Here, age and gender are the protected features. The labels are whether a person's credit is good or not.
- *Bank Marketing [11]:* This dataset consists of 45,211 instances and there are 17 features. Among them, age is the protected feature. The labels are whether the client will subscribe a term deposit or not.

We train three feed-forward neural networks following standard practice and run CARE to repair the unfair behavior of each model against the corresponding protected features.

For backdoor removal, CARE is evaluated over three datasets:

- *German Traffic Sign Benchmark Dataset (GTSRB) [60]:* This dataset consists of 39.2K training instances and 12.6K testing instances of colored images. The task is to recognize 43 different traffic signs. We train a CNN network consists of 6 convolutional layers and 2 dense layers.
- *MNIST [41]:* This dataset consists of 70K instances of handwritten digits as gray-scale images. We train a standard 4-layer CNN network to recognize the 10 digits (0-9).
- *Fashion-MNIST [69]:* This data set consists of 70K instances of 10 fashion categories, e.g., dress, coat and etc. Each sample is a 28x28 grayscale image. We train a CNN network consists of 3 convolutional layers and 2 dense layers.

For safety violation repairing, CARE is evaluated over three ACAS Xu [33] networks. ACAS Xu contains an array of 45 DNNs that produces horizontal maneuver advisories of unmanned version of Airborne Collision Avoidance System X. As discovered in [34, 50], some DNNs violates certain safety properties, e.g., DNN $N_{2,9}$ violates the safety property $\phi_8$. We apply CARE on 3 sub networks $N_{2,9}$, $N_{3,3}$ and $N_{1,9}$ against 3 properties $\phi_8$, $\phi_2$ and $\phi_7$ respectively, aiming to repair the misbehavior. Table 1 shows the details of the trained networks used in our experiment.

In PSO, we follow the general recommendation in [51] and set parameter $\omega = 0.8$, $c_1 = c_2 = 0.41$ and number of particles to 20. The maximum number of iteration is set to 100. To further reduce the searching time, we stop the search as soon as the property is satisfied or we fail to find a better location in the last 10 consecutive iterations. Note that CARE performs accuracy check after PSO finds an $M$. If the accuracy drop is significant, i.e., bigger than a threshold of 3%, CARE returns error message and a larger value of $\alpha$ shall be set in such scenarios.

**Table 1: Neutral Networks Used in Our Experiments**

| Model | Dataset | Architecture | #Neuron | Accuracy |
|---|---|---|---|---|
| $NN_1$ | Census | 6-layer FFNN | 139 | 0.8818 |
| $NN_2$ | Credit | 6-layer FFNN | 146 | 1.0 |
| $NN_3$ | Bank | 6-layer FFNN | 143 | 0.9226 |
| $NN_4$ | GTSRB | 6-Conv + 2 Dense CNN | 107,595 | 0.9657 |
| $NN_5$ | MNIST | 2-Conv + 2 Dense CNN | 31,610 | 0.9909 |
| $NN_6$ | Fashion | 3-Conv + 2 Dense CNN | 67,226 | 0.9136 |
| $NN_7$ | ACAS $N_{2,9}$ | 6-layer FFNN | 305 | - |
| $NN_8$ | ACAS $N_{3,3}$ | 6-layer FFNN | 305 | - |
| $NN_9$ | ACAS $N_{1,9}$ | 6-layer FFNN | 305 | - |

## 4.2 Research Questions and Answers

In the following, we report our experiment results and answer multiple research questions.

*RQ1: Is CARE successful in neural network repair?* To answer this question, we systematically apply CARE to the above-mentioned neural networks.

For *fairness repair tasks*, to better evaluate the performance of CARE, we set a strict fairness requirement (i.e., $\xi = 1\%$) to make sure all models fail the fairness property and CARE performs fairness repair on all of them (note that some of models fail fairness even if $\xi$ is set to be a realistic value of 5%). Table 2 summarizes the results, where the columns show model, the protected feature, the unfairness (maximum probability difference) before and after repair and model accuracy before and after repair. Note that in this experiment, input features are assumed to be independent and normal distribution is followed in our sampling process. The number of neurons to repair is set to be no more than 10% of total number of neurons in the given network. As illustrated in Table 2, among all cases, $NN_2$ and $NN_3$ show alarming fairness concern (i.e., with unfairness above 5%). CARE successfully repairs all neural networks, with an average fairness improvement of 61.9% and maximum of 99.2%. In terms of model accuracy, either the accuracy is unchanged ($NN_2$) or has only a slight drop ($NN_1$ and $NN_3$).

We further compare the performance of CARE with the state-of-the-art work [61] for this task. The method proposed in [61], relies on learning a Markov Chain from original network and performing sensitivity analysis on it. Then optimization is applied to find small adjustments to weight parameters of sensitive neurons for better fairness. We run experiments on $NN_1$, $NN_2$ and $NN_3$ and the performance comparison is shown in Table 5 (DTMC represents the method proposed in [61]). CARE is able to improve fairness by 61.9% on average and the model accuracy drops by 1.7%, while DTMC only improves fairness by 45.1% at a higher cost of 2.2%.

For *backdoor removal tasks*, we train neural networks $NN_4$, $NN_5$ and $NN_6$ following the attack methodology proposed in BadNets [26]. We randomly chose a target label for each network and vary the

ratio of adversarial inputs in training to achieve a high attack success rate of > 95% while maintaining high accuracy. The trigger is a white square located at the bottom right of the image with size $4 \times 4$ (around 1% of the entire image). In the experiment, we assume the trigger is unknown (as it is often the case in real application) and testing dataset is available. We adopt the technique proposed in [67] to reverse engineer the trigger. As discovered in [67], the reverse-engineered trigger is not perfect in terms of visual similarity. However, CARE is able to remove the backdoor effectively as shown below. Since $NN_4$, $NN_5$ and $NN_6$ are CNNs and convolutional layers tend to extract local features such as edges, textures, objects and scenes [70], we apply CARE on dense layers only. We measure the attack success rate (SR) by adding the reverse-engineered trigger to all images in the testing set. As shown in Table 3, for all the three models, CARE is able to mitigate the backdoor attack by reducing the attack SR from over 98% to less than 1%, while the accuracy of the repaired networks either maintains the same ($NN_4$) , reduced by < 1% ($NN_5$) or even improved ($NN_6$).

Furthermore, we compare CARE with the state-of-the-art neural network repair technique proposed in [65] named NNRepair. NNRepair leverages activation map [24] to perform fault localization of a buggy neural network and fix undesirable behaviors using constraint solving. We conduct experiments over CARE and NNRepair on two CNN models (experiment subjects of NNRepair) trained on MNIST [41] and CIFAR10 [39] datasets. The average performance comparison is shown in Figure 5. CARE is able to reduces the SR by 99.9% with accuracy drop of 1.5% on average. In contrast, NNRepair only reduce SR by 19.3% and model accuracy drops by 5.9%.

For *safety property repair tasks*, we use $N_{2,9}$, $N_{3,3}$ and $N_{1,9}$ of ACAS Xu networks as our $NN_7$, $NN_8$ and $NN_9$. These networks take 5-dimentional inputs of sensor readings and output 5 possible maneuver advises. Katz *et al.* [34] and Long *et al.* [50], demonstrate that $N_{2,9}$ violates the safety property $\phi_8$, $N_{3,3}$ violates property $\phi_2$ and $N_{1,9}$ violates property $\phi_7$. Therefore, we apply CARE on these 3 networks to improve their performance on the corresponding property. In this experiment, for each network, we randomly sample 10K counterexamples to the property as the evaluation set and 10K instances that are correctly classified by the original neural network as the drawdown set. We measure the violation rate (VR) on each set to evaluate the performance. As shown in Table 4, for all the three networks, CARE successfully brings down the violation rate from 1.0 to < 1% in evaluation set, while the performance in drawdown set is not affected.

To further evaluate the performance of CARE on safety property repair tasks, we compare CARE with the state-of-the-art approach proposed in [59] named PRDNN. PRDNN [59] introduces decoupled neural network architecture and solves neural network repair problem as a linear programming problem. We apply PRDNN on the above mentioned three ACAS Xu networks. PRDNN fails to repair $NN_{1,9}$ (program hangs) and the average result of $NN_{2,9}$ and $NN_{3,3}$ is shown in Table 5. Both tools are able improve the performance effectively while CARE outer-performs PRDNN by 5.5% at a lower cost.

Thus to answer RQ1, we say CARE is able to repair given buggy neural network successfully for various properties while maintaining high model accuracy. Furthermore, CARE outer-performs existing works for different tasks.

**Table 2: Fairness Repair Results**

| Model | P. Feat. | Fairness Score | | Accuracy | | Time(s) | |
|---|---|---|---|---|---|---|---|
| | | Before | After | Before | After | Loc | Tot |
| $NN_1$ | Race | 0.0431 | 0.0119 | 0.88 | 0.85 | 118 | 314 |
| $NN_1$ | Age | 0.0331 | 0.0230 | 0.88 | 0.86 | 173 | 276 |
| $NN_1$ | Gender | 0.0130 | 0.0070 | 0.88 | 0.86 | 120 | 240 |
| $NN_2$ | Age | 0.0659 | 0.0005 | 1.0 | 1.0 | 170 | 221 |
| $NN_2$ | Gender | 0.0524 | 0.0374 | 1.0 | 1.0 | 103 | 129 |
| $NN_3$ | Age | 0.0544 | 0.0028 | 0.92 | 0.91 | 174 | 435 |

*RQ2: What is the effect of parameter $\alpha$, i.e., the trade-off between the degree of property violation and accuracy?* Recall that the value of $\alpha$ controls the trade-off between fairness and accuracy. To understand the effect of $\alpha$'s value, we perform another set of experiments.

Figure 5 illustrates the result on all neural networks, where the first plot shows the performance improvement of repaired network compared to original network and the second plot shows the cost measured by model accuracy drop. As $\alpha$ increases from 0.1 to 0.9, the importance of model accuracy over repair objective in PSO fitness function increases. As shown in the plots for $NN_1$, the model accuracy is quite sensitive with respect to the value of $\alpha$, i.e., when $\alpha = 0.2$ model accuracy drops by 0.05 (from 0.88 to 0.83). Although a smaller $\alpha$ results in better fairness, we select a larger $\alpha$ to keep high model accuracy. As shown in the plots for $NN_2$, model accuracy is stable over different $\alpha$ values. Similar to the previous case, smaller $\alpha$ results in better fairness. Therefore, we select a small $\alpha$ for more effective fairness repair. As for $NN_3$, a large $\alpha$ improves model accuracy significantly. When $\alpha$ is greater than 0.7, the model accuracy is even higher than the original network. That is because PSO tries to optimize for model accuracy as well. However, the effectiveness of fairness improvement drops, i.e., model unfairness is reduced to 0.001 for $\alpha = 0.1$ but at $\alpha = 0.9$ the unfairness is as high as 0.0368. For $NN_4$, $NN_5$ and $NN_6$ costs for all $\alpha$ values are small (< 1%). This is because fixing the backdoor itself will improve model accuracy as well. For $NN_4$ and $NN_5$ the performance improvement (drop in SR) is quite stable over different $\alpha$. For $NN_6$, the performance improvement drops as $\alpha$ increases, hence, we select small $\alpha$ to let the optimization focus on backdoor removal. As for $NN_7$, $NN_8$ and $NN_9$, the performance improvement is stable over different $\alpha$'s values. For $NN_7$, the cost drops from 0.25% to 0.02% when $\alpha$ increases from 0.1 to > 0.2. While for $NN_8$ and $NN_9$, value of $\alpha$ does not affect the cost much. Hence, we select small $\alpha$ value for $NN_7$ and $NN_8$ and select $\alpha > 0.1$ for $NN_9$. As shown by the experiment results, the value of $\alpha$ balances the trade-off between performance improvement and cost. A smaller $\alpha$ often results in more effective property repair but model accuracy may be affected. In our experiments, we set the value of $\alpha$ as shown in Table 6.

Thus to answer RQ2, we say that $\alpha$ does have a significant impact on the repairing result. In practice, the users can decide its value based on the importance of the property. In the future, we aim to study systematic ways of identifying the best $\alpha$ value automatically.

**Table 3: Backdoor Removal Results**

| Model | Attack SR | | Accuracy | | Time(s) | |
|---|---|---|---|---|---|---|
| | Before | After | Before | After | Loc | Tot |
| $NN_4$ | 0.9808 | 0 | 0.9657 | 0.9654 | 80 | 493 |
| $NN_5$ | 0.9868 | 0 | 0.9909 | 0.9867 | 330 | 541 |
| $NN_6$ | 0.9977 | 0.0007 | 0.9136 | 0.9199 | 108 | 492 |

**Table 4: Safety Property Repair Results**

| Model | Counterexample VR | | Drawndown VR | | Time(s) | |
|---|---|---|---|---|---|---|
| | Before | After | Before | After | Loc | Tot |
| $NN_7$ | 1.0 | 0.0065 | 0.0 | 0.0 | 33 | 503 |
| $NN_8$ | 1.0 | 0.0 | 0.0 | 0.0 | 31 | 168 |
| $NN_9$ | 1.0 | 0.0014 | 0.0 | 0.0 | 34 | 276 |

**Table 5: Comparison with Existing Works**

| Tech. | Fairness | | Backdoor | | Safety | |
|---|---|---|---|---|---|---|
| | Imp. | Cost | Imp. | Cost | Imp. | Cost |
| CARE | 61.9% | 1.7% | 99.9% | 1.5% | 99.7% | 0 |
| DTMC | 45.1% | 2.2% | - | - | - | - |
| NNREPAIR | - | - | 19.3% | 5.9% | - | - |
| PRDNN | - | - | - | - | 94.1% | 0.01% |

**Table 6: Parameter $\alpha$ Setting**

| Model | $NN_1$ | $NN_1$ | $NN_1$ | $NN_2$ | $NN_2$ | $NN_3$ |
|---|---|---|---|---|---|---|
| P. Feat. | Race | Age | Gender | Age | Gender | Age |
| $\alpha$ | 0.1 | 0.7 | 0.8 | 0.1 | 0.1 | 0.3 |
| Model | $NN_4$ | $NN_5$ | $NN_6$ | $NN_7$ | $NN_8$ | $NN_9$ |
| $\alpha$ | 0.2 | 0.2 | 0.2 | 0.3 | 0.1 | 0.2 |

*RQ3: How effective is the causality-based fault localization?* This question asks whether our causality-based fault localization is useful. We answer this question by comparing the repair effectiveness through optimizing the weight parameters that are selected in four different ways, i.e., 1) selected based on our approach; 2) selected randomly; 3) selected based gradients and 4) include all the parameters. The results are shown in Table 7.

Firstly, we perform network repair with randomly selected neurons to optimize. We follow the same configuration of CARE as the one we used in RQ1, i.e., with the same $\alpha$ value and the number of neurons to fix. The performance improvement and cost of the repair is shown in *Random1* columns in Table 7. Adjusting randomly selected parameters results in an average performance improvement of 47.6%. While CARE improves the performance by 80.8% on all tasks. For fairness repair tasks, the improvement is significant with randomly selected neurons. However, the model accuracy drops sharply. The overall accuracy drop is 8.9% on average with the worst case of 19.0% for $NN_2$ w.r.t. protected feature gender. For backdoor

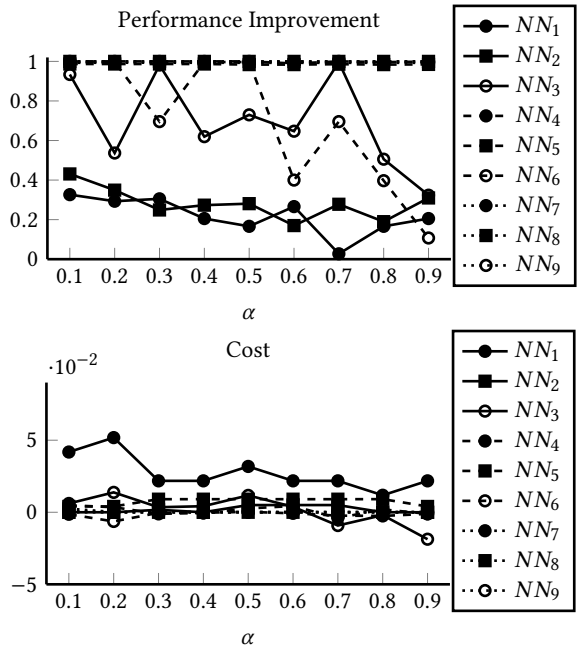


**Figure 5: Effect of Parameter $\alpha$**

removal tasks, the performance improvement is 6.4% on average and model accuracy is affected. Especially for $NN_6$, accuracy is reduced by 6.3% after the repair. In terms of safety repair tasks, the repair is not so effective. For $NN_7$ the performance improvement is less than 1% and in average the improvement is only 36.6%, although the model accuracy is not affected much. Therefore, with randomly selected parameters, the repair is not effective and the performance is improved at the cost of disrupting correct behavior of the original model. On the other side, as described in Section 3, our fault localization is based on the causal influence of each neuron on the targeted misbehavior without altering other behaviors.

Secondly, we optimize all weight parameters of the given neural network, i.e., no fault localization. In this setting, the search space in the PSO algorithm increases significantly (compared to the case of repairing 10% of parameters in CARE). Therefore, we limit the time taken by PSO so that time allowed to spend in this step is the same as that taken by CARE with fault localization. The results are shown in columns *All*. For fairness repair tasks the fairness is improved by 67.8% on average. But the model accuracy drops significantly, i.e., the accuracy drops by 20.1% on average and 42.0% in the worst case. For backdoor removal, the performance improvement is only 36.2% on average, while CARE reduces the SR by 99.5%. For safety repair tasks, the performance improvement is 55.6% on average and CARE manages to reduce the VR in evaluation set by 99.8%. Although the cost is below 1% for these two tasks, the repair is not that effective. In all of the cases without fault localization, PSO is not able to converge within the time limit and as a result, neither performance improvement nor cost is optimized when PSO terminates.

In the literature, gradient is a commonly used technique that guides the search in program fault localization, program repair and fairness testing [20, 57, 72]. Therefore, we conduct experiments

| Model | Performance Improvement | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | CARE | Gradient | Random | All | CARE | Gradient | Random | All |
| $NN_1$ Race | 0.7222 | 0.9953 | 0.9953 | 1.0 | 0.0318 | 0.1218 | 0.1418 | 0.1318 |
| $NN_1$ Age | 0.3051 | 0.9486 | 0.9987 | 0.9848 | 0.0218 | 0.1818 | 0.1018 | 0.1418 |
| $NN_1$ Gender | 0.4615 | 0.6692 | 0.7384 | 0.9230 | 0.0218 | 0.1418 | 0.1118 | 0.1018 |
| $NN_2$ Age | 0.9924 | 0.1365 | 0.4324 | 0.0273 | 0.0 | 0.4000 | 0.1200 | 0.3500 |
| $NN_2$ Gender | 0.2847 | 0.7345 | 0.5989 | 0.1348 | 0.0 | 0.2000 | 0.1900 | 0.4200 |
| $NN_3$ Age | 0.9485 | 0.9301 | 0.6562 | 1.0 | 0.0125 | 0.0226 | 0.0426 | 0.0626 |
| $NN_4$ | 1.0 | 0.0174 | 0.0050 | 0.3901 | 0.0002 | 0.0032 | 0.0050 | 0.0056 |
| $NN_5$ | 0.9868 | 0.1058 | 0.0938 | 0.5220 | 0.0040 | 0.0019 | 0.0 | 0.0038 |
| $NN_6$ | 0.9970 | 0.6995 | 0.0935 | 0.1729 | −0.006 | 0.0016 | 0.0628 | 0.0034 |
| $NN_7$ | 0.9935 | 0.0004 | 0.0207 | 0.5707 | 0.0 | 0.0 | 0.0 | 0.0001 |
| $NN_8$ | 1.0 | 0.9250 | 0.6440 | 0.7485 | 0.0 | 0.0 | 0.0 | 0.0 |
| $NN_9$ | 1.0 | 0.9933 | 0.4308 | 0.3478 | 0.0 | 0.0 | 0.0 | 0.0008 |

**Table 7: Fault Localization Effectiveness**

to compare the performance of CARE with gradient-guided fault localization. That is, instead of calculating causal attribution of the hidden neurons, the fault localization step is based on $\frac{\partial y}{\partial v}$ where $y$ represents the favourable output value and $v$ represents hidden neuron value. We use the gradient-based method to identify the same amount of "responsible" neurons and perform optimization with the same setting as CARE. The results of gradient-based localization method is illustrated in columns *Gradient* of Table 7. Compared with CARE, for fairness repair tasks, the gradient-based method is able to find a repair that satisfies fairness criteria but fails to maintain model accuracy. Overall fairness improves by 73.6% but average accuracy drops by 17.8%, which is not desirable. For backdoor removal, the gradient-based method is not able to find an effective repair where the overall performance improvement is below 30%. Especially for $NN_4$, the SR is still as high as 96.4% after the repair. For safety repair, gradient-based method is able to find a repair for $NN_8$ and $NN_9$ with performance improvement above 90%, while CARE archives 100% backdoor removal. For $NN_7$, gradient-based method is not useful at all where VR in evaluation set is as high as 99% after the repair. Hence the performance of gradient-based method is not stable and often not effective.

Thus to answer RQ3, we say our causality-based fault localization is effective in identifying candidates for parameter optimization. It guides CARE to focus on fixing undesirable behavior of the model while keeping the correct behavior unaffected.

## 5 RELATED WORK

This work is broadly related to works on neural network verification, repair and causal interpretation.

*Neural network verification.* There have been an impressive line of methods proposed recently for neural network verification. These includes solving the problem using abstraction techniques [16, 22, 56], SMT sovler [30, 34, 35], MILP and LP [15, 62], symbolic execution [68] and many others [7, 14, 25, 37]. There have also been attempts to verify neural network fairness properties, including [4] and [5] based on numerical integration , [31] based on non-convex optimization and [61] based on probabilistic model checking. Unlike these works, our approach focus on neural network repair instead and we leverage some existing verification methods [59, 61, 67] in our work for property verification.

*Machine learning model repair.* There have been multiple attempts on repairing machine learning models to remove undesirable behaviors. In [36], Kauschke *et al.* suggest a method to learn a model to determine error regions of the given model, which allows users to patch the given model. In [58], Sotoudeh *et al.* propose a method to correct a neural network by applying small changes to model weights based on SMT formulation of the problem. In [23], Goldberger *et al.* leverage recent advances in neural network verification and presented an approach to repair a network with minimal change in its weights. This approach aims to find minimal layer-wise fixes for a given point-wise specification and the performance is restricted by the underlying verification method used. NNRepair [65] performs constraint-based repair on neural networks to fix undesirable behaviors but only applies to a single layer. Sotoudeh *et al.* proposed a method in [59] to solve a neural network repair problem as a linear programming problem. Unlike our approach, both NNrepair and the method proposed in [59] perform layer-wise repair but the actual fault in a buggy network could be across multiple layers. Furthermore, based on our experiment results, our approach is more effective in neural network repair against different properties.

*Machine learning causal interpretation.* Causality analysis has been applied to generate explanations for machine learning algorithms. Existing works focus on causal interpretable models that can explain why machine learning models make certain decision. Narendra *et al.* [46] model DNNs as SCMs and estimate the causal effect of each component of the model on the output. In [8], Chattophadhyay *et al.* proposed an scalable causal approach to estimate individual causal effect of each feature on the model output. Causal inference has been applied in machine learning model fairness studies. Kusner *et al.* [40] proposed an approach to measure the fairness of a machine learning model based on counterfactuals where a fair model should

have the same prediction for both actual sample and counterfactual sample. In [71], Zhang *et al.* propose a causal explanation based metric to quantitatively measure the fairness of an algorithm. Our work utilizes SCMs and $do(\cdot)$ calculus to measure the causal attribution of hidden neurons on model undesirable behaviors. The results are used as a guideline for fault localization.

## 6 CONCLUSION

We present CARE, a causality-based technique for repairing neural networks for various properties. CARE performs fault localization on a given neural network model and utilizes PSO to adjust the weight parameters of the identified neurons. CARE generates repaired networks with improved performance over specified property while maintaining the model's accuracy. CARE is evaluated empirically with multiple neural networks trained on benchmark datasets and experiment results show that CARE is able to repair buggy model efficiently and effectively, with minimally disruption on existing correct behavior.

## 7 DATA AVAILABILITY

A Prove of Concept (PoC) realization of this work (CARE) is implemented on top of SOCRATES [50] as a causality-based neural network repair engine. The source code of CARE is publicly available at [50].

## 8 ACKNOWLEDGEMENTS

We thank anonymous reviewers for their constructive feedback. This research is supported by Huawei International (Grant Number TC20210714014).

## REFERENCES

[1] Jason Abrevaya, Yu-Chin Hsu, and Robert P Lieli. Estimating conditional average treatment effects. *Journal of Business & Economic Statistics*, 33(4):485–505, 2015.
[2] Aniya Agarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. Automated test generation to detect individual discrimination in AI models. *CoRR*, 2018.
[3] Aws Albarghouthi, Loris D'Antoni, and Samuel Drews. Repairing decision-making programs under uncertainty. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2017.
[4] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. Fairsquare: probabilistic verification of program fairness. *Proc. ACM Program. Lang.*, 1(OOPSLA):80:1–80:30, 2017.
[5] Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. Probabilistic verification of fairness properties via concentration. *PACMPL*, 3(OOPSLA):118:1–118:27, 2019.
[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, 2016.
[7] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Branch and bound for piecewise linear neural network verification. *J. Mach. Learn. Res.*, 21:42:1–42:39, 2020.
[8] Aditya Chattopadhyay, Piyushi Manupriya, Anirban Sarkar, and Vineeth N. Balasubramanian. Neural network attributions: A causal perspective. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 981–990. PMLR, 2019.
[9] Hana Chockler, Joseph Y. Halpern, and Orna Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Log.*, 9(3):20:1–20:26, 2008.
[10] Guoliang Dong, Jun Sun, Jingyi Wang, Xinyu Wang, and Ting Dai. Towards repairing neural networks correctly. *CoRR*, abs/2012.01872, 2020.
[11] Dheeru Dua and Casey Graff. Bank marketing dataset at UCI machine learning repository, 2017.
[12] Dheeru Dua and Casey Graff. Census income dataset at UCI machine learning repository, 2017.
[13] Dheeru Dua and Casey Graff. German credit dataset at UCI machine learning repository, 2017.
[14] Krishnamurthy (Dj) Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, and Pushmeet Kohli. Efficient neural network verification with exactness characterization. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, volume 115 of *Proceedings of Machine Learning Research*, pages 497–507. AUAI Press, 2019.
[15] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *CoRR*, abs/1705.01320, 2017.
[16] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 43–65. Springer, 2020.
[17] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia*, pages 259–268, 2015.
[18] Ricky Fok, Aijun An, and Xiaogang Wang. Optimization assisted MCMC. *CoRR*, abs/1709.02888, 2017.
[19] Kang Fu, Dawei Cheng, Yi Tu, and Liqing Zhang. Credit card fraud detection using convolutional neural networks. In *Neural Information Processing - 23rd International Conference (ICONIP 2016), Kyoto, Japan*, pages 483–490, 2016.
[20] Xin Gao, Fang Deng, and Xianghu Yue. Data augmentation in fault diagnosis based on the wasserstein generative adversarial network with gradient penalty. *Neurocomputing*, 396:487–494, 2020.
[21] Sahaj Garg, Vincent Perot, Nicole Limtiaco, Ankur Taly, Ed H. Chi, and Alex Beutel. Counterfactual fairness in text classification through robustness. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society (AIES 2019), Honolulu, HI, USA*, pages 219–226, 2019.
[22] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 3–18. IEEE Computer Society, 2018.
[23] Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. Minimal modifications of deep neural networks using verification. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 260–278. EasyChair, 2020.
[24] Divya Gopinath, Hayes Converse, Corina S. Pasareanu, and Ankur Taly. Property inference for deep neural networks. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 797–809. IEEE, 2019.
[25] Dennis Gross, Nils Jansen, Guillermo A. Pérez, and Stephan Raaijmakers. Robustness verification for classifier ensembles. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 271–287. Springer, 2020.
[26] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.
[27] Michael Harradon, Jeff Druce, and Brian E. Ruttenberg. Causal learning and explanation of deep neural networks via autoencoded activations. *CoRR*, abs/1802.00541, 2018.
[28] Amjad Ibrahim, Tobias Klesel, Ehsan Zibaei, Severin Kacianka, and Alexander Pretschner. Actual causality canvas: A general framework for explanation-based socio-technical constructs. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2978–2985. IOS Press, 2020.
[29] Kosuke Imai, Gary King, and Elizabeth Stuart. Misunderstandings among experimentalists and observationalists about causal inference. *Journal of the Royal Statistical Society, Series A*, 171, part 2:481–502, 2008.
[30] Yuval Jacoby, Clark W. Barrett, and Guy Katz. Verifying recurrent neural networks using invariant inference. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 57–74. Springer, 2020.
[31] Philips George John, Deepak Vijaykeerthy, and Diptikalyan Saha. Verifying individual fairness in machine learning models. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of the Thirty-Sixth Conference on Uncertainty in*

*Artificial Intelligence, UAI 2020, virtual online, August 3-6, 2020*, volume 124 of *Proceedings of Machine Learning Research*, pages 749–758. AUAI Press, 2020.

[32] Brittany Johnson, Yuriy Brun, and Alexandra Meliou. Causal testing: understanding defects' root causes. In Gregg Rothermel and Doo-Hwan Bae, editors, *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 87–99. ACM, 2020.

[33] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. Deep neural network compression for aircraft collision avoidance systems. *CoRR*, abs/1810.04240, 2018.

[34] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017.

[35] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 2019.

[36] Sebastian Kauschke and Johannes Fürnkranz. Batchwise patching of classifiers. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3374–3381. AAAI Press, 2018.

[37] Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. POPQORN: quantifying robustness of recurrent neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3468–3477. PMLR, 2019.

[38] Murat Kocaoglu, Christopher Snyder, Alexandros G. Dimakis, and Sriram Vishwanath. Causalgan: Learning causal implicit generative models with adversarial training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[39] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[40] Matt J. Kusner, Joshua R. Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4066–4076, 2017.

[41] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[42] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[43] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. Deepmutation: Mutation testing of deep learning systems. In *29th International Symposium on Software Reliability Engineering*, pages 100–111, 2018.

[44] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. MODE: automated neural network model debugging via state differential analysis and input selection. In Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu, editors, *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, pages 175–186. ACM, 2018.

[45] Álvaro Parafita Martínez and Jordi Vitrià Marca. Explaining visual models by causal attribution. In *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*, pages 4167–4175. IEEE, 2019.

[46] Tanmayee Narendra, Anush Sankaran, Deepak Vijaykeerthy, and Senthil Mani. Explaining deep learning models using causal inference. *CoRR*, abs/1811.04376, 2018.

[47] High-Level Expert Group on Artificial Intelligence (AI HLEG). Draft ethics guidelines for trustworthy ai. Technical report, European Commission, 2018.

[48] Judea Pearl. Cambridge university press, 2009.

[49] Judea Pearl. The seven tools of causal inference, with reflections on machine learning. *Commun. ACM*, 62(3):54–60, 2019.

[50] Long H. Pham, Jiaying Li, and Jun Sun. Socrates: Towards a unified platform for neural network verification. *ArXiv*, abs/2007.11206, 2020.

[51] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1:33–57, 2007.

[52] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intell.*, 1(1):33–57, 2007.

[53] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), Boston, MA, USA*, pages 815–823, 2015.

[54] Shahid Shabir and Ruchi Singla. A comparative study of genetic algorithm and the particle swarm optimization. *Int. J. Electr. Eng*, 9(2):215–223, 2016.

[55] Y. Shi and R. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming*, 1998.

[56] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, 2019.

[57] Jeongju Sohn, Sungmin Kang, and Shin Yoo. Search based repair of deep neural networks. *CoRR*, abs/1912.12463, 2019.

[58] Matthew Sotoudeh and A. Thakur. Correcting deep neural networks with small, generalizing patches. In *Workshop on Safety and Robustness in Decision Making*, 2019.

[59] Matthew Sotoudeh and Aditya V. Thakur. Provable repair of deep neural networks. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 20211*, pages 588–603. ACM, 2021.

[60] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.

[61] Bing Sun, Jun Sun, Ting Dai, and Lijun Zhang. Probabilistic verification of neural networks against group fairness. In Marieke Huisman, Corina S. Pasareanu, and Naijun Zhan, editors, *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings*, volume 13047 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2021.

[62] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[63] Florian Tramèr, Vaggelis Atlidakis, Roxana Geambasu, Daniel J. Hsu, Jean-Pierre Hubaux, Mathias Humbert, Ari Juels, and Huang Lin. Fairtest: Discovering unwarranted associations in data-driven applications. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P 2017), Paris, France*, pages 401–416, 2017.

[64] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. Automated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018), Montpellier, France*, pages 98–108, 2018.

[65] Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, and Corina S. Pasareanu. Nnrepair: Constraint-based repair of neural network classifiers. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, volume 12759 of *Lecture Notes in Computer Science*, pages 3–25. Springer, 2021.

[66] Sandra Vieira, Walter H.L. Pinaya, and Andrea Mechelli. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience & Biobehavioral Reviews*, 74:58–75, 2017.

[67] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 707–723. IEEE, 2019.

[68] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1599–1614. USENIX Association, 2018.

[69] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[70] Bin Yang, Honglei Guo, and Enguo Cao. Chapter two - design of cyber-physical-social systems with forensic-awareness based on deep learning. In Ali R. Hurson and Sheng Wu, editors, *AI and Cloud Computing*, volume 120 of *Advances in Computers*, pages 39–79. Elsevier, 2021.

[71] Junzhe Zhang and Elias Bareinboim. Fairness in decision-making - the causal explanation formula. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2037–2045. AAAI Press, 2018.

[72] P. Zhang, J. Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, J. Dong, and Dai Ting. White-box fairness testing through adversarial sampling. pages 1–12, 2020.

[73] Qingyuan Zhao and Trevor Hastie. Causal interpretations of black-box models. *Journal of Business & Economic Statistics*, 39(1):272–281, 2021.