Repairing Misclassifications in Neural Networks Using Limited Data

Patrick Henriksen*
Imperial College London
London, United Kingdom
patrick.henriksen18@imperial.ac.uk

Francesco Leofante*
Imperial College London
London, United Kingdom
f.leofante@imperial.ac.uk

Alessio Lomuscio
Imperial College London
London, United Kingdom
a.lomuscio@imperial.ac.uk

ABSTRACT

We present a novel and computationally efficient method for repairing a feed-forward neural network with respect to a finite set of inputs that are misclassified. The method assumes no access to the training set. We present a formal characterisation for repairing the neural network and study its resulting properties in terms of soundness and minimality. We introduce a gradient-based algorithm that performs localised modifications to the network's weights such that misclassifications are repaired while marginally affecting network accuracy on correctly classified inputs. We introduce an implementation, I-REPAIR, and show it is able to repair neural networks while reducing accuracy drops by up to 90% when compared to other state-of-the-art approaches for repair.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; Artificial intelligence; • **Software and its engineering** → **Software testing and debugging**;

KEYWORDS

Model Repair, Deep Neural Networks, Safe AI

ACM Reference Format:
Patrick Henriksen, Francesco Leofante, and Alessio Lomuscio. 2022. Repairing Misclassifications in Neural Networks, Using Limited Data. In *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event, .* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3477314.3507059

1 INTRODUCTION

Neural Networks (NNs) have demonstrated excellent performance in several domains within computer science [14] and are often adopted in industrial applications. Given the high computational requirements that are often needed to train state-of-the-art NNs, a new industrial practice has emerged where vendors ship pre-trained models to end-users that can use them for inference in their specific applications [32].

Yet, by the generic nature of these models, end-users often need to tailor them to their specific needs and operating conditions. Indeed, testing [37, 38], verification [1–5, 7, 11, 17–19, 23, 24, 26, 28, 34, 39, 41–43] or runtime analysis [8–10, 20] of these models may reveal deficiencies, or misclassifications, which may have severe consequences if not addressed.

However, repairing such deficiencies without access to the original training data introduces fundamental challenges as standard, data-intensive, training methods cannot be applied. In the setting we consider here, the end-user has only access to smaller batches of correct classifications obtained, e.g., by storing manually-annotated runtime predictions made by the network once deployed. Leveraging such limited data, repair operations should fix detected misclassifications without having a significant impact on the previously measured accuracy. However, only few approaches are available to repair neural networks under these assumptions.

Contribution. In this paper we put forward an approach to solve the challenge above. We consider the problem of repairing a pre-trained NN against a finite set of misclassifications without requiring access to the original training set. The method we develop aims to adjust parameters in the network in a computationally efficient way such that misclassifications can be corrected while heuristically minimising the impact that repair interventions may have on network accuracy. Concretely, we make the following contributions:

- We study the theoretical limitations of repair; discuss how existing repair metrics may lead to an uncontrolled loss of accuracy in repaired networks and put forward a new repair metric that directly accounts for network accuracy.
- We propose an approach to identify the parameters on the model that should be altered to repair misclassifications without severely compromising its accuracy.
- We develop an algorithm for repair and empirically show that it outperforms state-of-the-art methods based on weight-minimality as well as naïve retraining in the limited-data setting.

The rest of the paper is organised as follows. After discussing related work below, in Section 2 we introduce the necessary background on NNs and related concepts. In Section 3 we formulate our target concept of repair formally and explore some resulting properties. Notably, this includes an impossibility result for sound repair in the general setting. Section 4 introduces an algorithm to compute influence, a measure of impact of network parameters with respect to particular inputs, and the repair procedure itself which we analyse in terms of correctness and minimality. Section 5 reports details of an implementation as well as the experimental

*Equal Contribution

SAC'22, April 25–April 29, 2022, Brno, Czech Republic

ACM ISBN 978-1-4503-8713-2/22/04…$15.00
https://doi.org/10.1145/3477314.3507059

results obtained on fully-connected and ResNet [16] architectures. We conclude in Section 6.

Related Work. Several approaches have been proposed in recent years to address the repair problem [13, 15, 30, 35, 36, 40, 44]

Relevant to this contribution are those approaches that frame repair as the problem of finding minimal modifications to weights of a learned model. For instance, [15] deal with minimal repair of Kernel Ridge Regression with Random Fourier Features by casting this as an optimisation problem, which is then solved using off-the-self optimisation tools. As noted in [13], this approach is not applicable to NNs since the resulting optimisation problem would be highly non-convex and high-dimensional. In [13] NN verification methods are used to produce repair operations that are provably minimal and do not require additional data nor access to the training set. This approach, however, suffers from severe scalability limitations as proving minimality is hard in general. To overcome this limitation, the authors restrict repair operations only to the last layer of a NN; however, even then, scalability remains an issue. The resulting procedure operates successfully on a network with one hidden layer of 150 nodes but fails to terminate on deeper models.

Closely related to this work is [30], which presents MODE, a technique for locating and repairing bugs in NNs. Similarly to our proposal, the authors develop a lightweight technique to identify model parameters (i.e., weights) that are critical for a misclassification; however, differently from us, the authors consider a parameter to be influential based on its magnitude, not its gradient. This is a crucial difference: if repair aims to minimise its impact on the overall NN performance, the sensitivity measure provided by gradients should be taken into account. The importance of considering gradients is also recognised in [35], where the authors put forward Arachne, a repair procedure that uses gradients to identify parameters to be fixed and uses Particle Swarm Optimisation to this end. Differently from us, however, Arachne constrains repair operation to happen in the last layer only and is therefore less flexible.

[40] recently proposed NNRepair, another technique to locate faulty network parameters aiming to perform minimal changes to the original NN. Their localisation phase starts by storing the activation patterns of neurons in a NN for a large number of inputs; decision-tree learning is then applied over such patterns to learn activation patterns that reflect valid NN behaviours. The experiments reported for a MNIST network in [40] reveal that this step requires ~6000 input *per class*, which makes this approach inapplicable when only limited data is available.

We conclude by mentioning that counterexamples identified via verification have been used to improve the accuracy of NNs [12, 25, 31] via retraining. However, these methods require the user to have access to the training set and do not provide any measure of the actual changes carried out in the network after retraining.

2 BACKGROUND AND NOTATION

Neural networks. A neural network (NN) is a learning model characterised by multiple levels of representation $\mathbf{h}^{(i)}$, $i \in \{1, \ldots, n\}$ obtained by composing (typically non-linear) computational units [14]. Starting from an input image $\mathbf{x}$, each unit transforms the representation of the preceding level into the next, until a target output representation $\mathbf{y}$ is obtained.

Given an input representation $\mathbf{h}^{(i-1)}$, a *linear unit* computes a linear combination of the values in $\mathbf{h}^{(i-1)}$; a *convolutional unit* computes the convolution between $\mathbf{h}^{(i-1)}$ and a learned kernel; and a *batch-norm unit* that applies a normalisation to $\mathbf{h}^{(i-1)}$ through re-centering and rescaling [22]. We use $\mathbf{L}^{(i)}$, $\mathbf{C}^{(i)}$ and $\mathbf{BN}^{(i)}$ to denote transformations computed by linear, convolutional and batch-norm units respectively.

Given an input $\mathbf{h}^{(0)} = \mathbf{x}$, a *fully-connected* NN $\mathcal{N}$ recursively applies a transformation $\mathbf{h}^{(i)} = \mathbf{\Phi}^{(i)}(\mathbf{L}^{(i)}(\mathbf{h}^{(i-1)}))$, $i = 1, \ldots, n-1$ and produces an output $\mathbf{y} = \mathcal{N}(\mathbf{x}) = \mathbf{L}^{(n)}(\mathbf{h}^{(n-1)})$, where $\mathbf{\Phi}^{(i)}$ is an *activation function* associated with the linear unit. In this work we consider NNs using *Rectified Linear Unit* activations $\mathrm{ReLU}(z) = \max(0, z)$. A *convolutional* NN can be obtained by redefining $\mathbf{h}^{(i)} = \mathbf{\Phi}^{(i)}(\mathbf{BN}^{(i)}(\mathbf{C}^{(i)}(\mathbf{h}^{(i-1)})))$.

Classification tasks. When used for classification, $\mathcal{N}$ is trained on a set of images drawn from an input space $\mathcal{S} \subset \mathbb{R}^n$. Each image is associated with a *label* generated by an unknown function $L : \mathcal{S} \to \{1, \ldots, s\}$, where $\{1, \ldots, s\}$ is the set of possible labels. Given an input $\mathbf{x}$, we use $\mathcal{N}(\mathbf{x})_c$, $c \in \{1, \ldots, s\}$ to denote the score computed by $\mathcal{N}$ for class c. The *prediction* of $\mathcal{N}$ for input $\mathbf{x}$ is computed as $\arg\max_{c \in \{1,\ldots,s\}} \mathcal{N}(\mathbf{x})_c$. Given a labelled training set, the goal of training is to learn a classifier $\mathcal{N}$ that can be used for inference: given a new image drawn from the same distribution, $\mathcal{N}$ should predict its true label. We refer to [14] for more details.

3 REQUIREMENTS FOR REPAIR

In this Section we formalise some key notions on repair. Specifically, we are interested in repair operations that modify the parameters of a pre-trained NN so that the resulting ("repaired") model satisfies certain properties.

Definition 1 (Repair operation P). *A* repair operation *is a function* P *mapping a NN $\mathcal{N}$ into a repaired network $\mathcal{N}^*$ such that $\mathcal{N}$ and $\mathcal{N}^*$ differ only in the values of their weights and bias.*

In the following, we will consider networks that are correct with respect to a particular set of correctly classified inputs, but incorrect with respect to another set, which we intend to repair.

Definition 2 (Correct set C and repair set $\mathcal{R}$). *Consider a NN $\mathcal{N}$, an input space $\mathcal{S} \subset \mathbb{R}^n$ and a labelling function $L : \mathcal{S} \to \{1, \ldots, s\}$. A* correct set C *is a set of pairs $(\mathbf{x}, l)$, where $x \in \mathcal{S}$ and $l = L(\mathbf{x})$, such that for all $(\mathbf{x}, l) \in C$, we have $\mathcal{N}(\mathbf{x}) = l$. A* repair set $\mathcal{R}$ *is a set of pairs $(\mathbf{x}, l)$ as above such that for all $(\mathbf{x}, l) \in \mathcal{R}$, we have $\mathcal{N}(\mathbf{x}) \neq l$.*

The set C can be seen as a proxy for regions of the input space $\mathcal{S}$ where the decision of the NN are correct, while $\mathcal{R}$ is a proxy for regions where decisions need repairing. Intuitively, larger sets result in better approximations of the decision-making of the NN; however here we assume these sets to be gathered at runtime and thus to contain few elements. For notational convenience, we sometimes refer to inputs in C (resp., $\mathcal{R}$) as $\mathcal{X}_C$ and labels as ψ_C (resp., $\mathcal{X}_\mathcal{R}, \psi_\mathcal{R}$).

Ideally, a *sound* repair operation should find modifications to the weights such that the network's response to inputs in $\mathcal{R}$ is corrected without altering the behaviour on inputs in C.

Definition 3 (Soundness of P). *Consider a NN $\mathcal{N}$, a correct set C, a repair set $\mathcal{R}$ and a repair operation* P. *Let $\mathcal{N}^*$ be a repaired network obtained via* P. *The repair operation* P *is* sound *iff for all $(\mathbf{x}, l) \in C$ we have that $\mathcal{N}^*(\mathbf{x}) = l$ and for all $(\mathbf{x}, l) \in \mathcal{R}$, we have that $\mathcal{N}^*(\mathbf{x}) = l$.*

We note that sound repair is generally impossible.

Theorem 1 (Existence of sound P). *Given any network $\mathcal{N}$ with ReLU activation functions, there are correct sets and repair sets for which no sound repair operation exists.*

Proof. A ReLU network with fixed topology has a bounded number b of linear regions [33], thus any $C, \mathcal{R}$ that require more linear regions than b cannot be represented by the network. □

Since soundness cannot be guaranteed a priori, a repair operation will in general need to strike a balance between correcting as many erroneous inputs as possible, and minimising any changes in the correct set. Previous work [13] proposed to account for this trade-off by suggesting that repairing a network would consist in searching the space of minimally-different models to find good repair candidates.

Definition 4 (Minimality of P). *Consider a NN $\mathcal{N}$, a repair set $\mathcal{R}$ and a repair operation* P. *Let $\mathcal{N}^*$ be a repaired network obtained via* P. *The repair operation is* minimal *iff: (i) for all $(\mathbf{x}, l) \in \mathcal{R}, \mathcal{N}^*(\mathbf{x}) = l$ and (ii) $d(\mathcal{N}^*, \mathcal{N}) \leq d(\mathcal{N}', \mathcal{N})$ for some metric d and all possible models $\mathcal{N}'$ obtained via any repair operation defined as per Definition 1.*

The definition above leaves the metric d open, as several metrics could be, and indeed have been, used. For instance, the key concept of L-distance defined in [13] can be seen in line with Def. 4 by defining d to be the L-norm of the difference between the two networks' weights. In this setting, two models are assumed to behave similarly if the difference between their weights is bounded. Repair should therefore apply minimal modifications to the weights in an attempt to fix the network on inputs in $\mathcal{R}$, under the assumption that these modifications will not affect inputs in C.

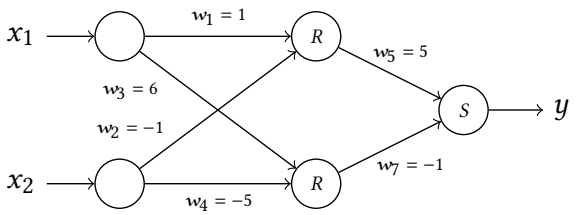
While this approach may be intuitive, it does not guarantee that elements of C will be classified in the same way following repair and may, in general, result in an uncontrolled loss of accuracy in the repaired network as we will show experimentally in Section 5. We show an instance of this problem in Example 1 below.

Example 1. *Consider the NN below with inputs (x_1, x_2), one hidden layer with two nodes with ReLU activations (R) and one output node with binary step activation (S).*

Let $\mathcal{R} = \{((1, 0), 1)\}$ and $C = \{((1, 1), 0)\}$. As given, the output of both the repair and correct inputs is 0.

A repair that is small with respect to weight change can be achieved by setting $w_1 = 1.4$. With this modification, the output of the repair input $(1, 0)$ is now fixed and becomes 1*; however, at the same time, the output of the correct input is also affected and becomes* 1.

An alternative solution is to set $w_5 = 7$; this results in the repair input attaining its desired output while leaving the output of the correct input unchanged. So, even though the magnitude of the weight change is larger, this repair is preferable.



We find that occurrences such as those above are widespread on large networks. In a nutshell, by minimally modifying weights we may still cause the misclassification of other elements. To overcome this issue, we propose using a different metric which directly captures the network's behavioural responses. A natural choice in this context is to consider the model accuracy.

Definition 5 (Accuracy wrt a set). *Let $\mathcal{N}$ be a NN and let $\mathcal{D}$ be a set of pairs $(\mathbf{x}_i, l_i)$, where l_i is the true label associated with $\mathbf{x}_i$ for $i = 1, \ldots, m$. The* accuracy of $\mathcal{N}$ with respect to $\mathcal{D}$ *can be defined as $\alpha(\mathcal{N}, \mathcal{D}) := \frac{1}{m} \cdot \sum_{i=1}^{m} \mathbb{1}(\arg\max_c \mathcal{N}(\mathbf{x}_i)_c = l_i)$.*

By considering accuracy one can reason explicitly over the response of the NN to a given input; this information can then be used to guide repair operations. We use these considerations to define a novel repair metric and amend the notion of minimality to quantify, and account for, the effects that repair has on the accuracy with respect to the set C.

Definition 6 (Accuracy drop induced by repair). *Consider a NN $\mathcal{N}$ and a set $\mathcal{D}$ as per Definition 5. Let $\alpha(\mathcal{N}, \mathcal{D})$ be the accuracy of $\mathcal{N}$ wrt $\mathcal{D}$. Now consider a repair operation* P *and let $\mathcal{N}^*$ be the network obtained by applying* P *to $\mathcal{N}$. We define the* accuracy drop induced by repair as $d(\mathcal{N}, \mathcal{N}^*, \mathcal{D}) = \alpha(\mathcal{N}, \mathcal{D}) - \alpha(\mathcal{N}^*, \mathcal{D})$.

Definition 7 (Minimality of P wrt C). *Consider a NN $\mathcal{N}$, a correct set C, a repair set $\mathcal{R}$ and a repair operation* P. *Let $\mathcal{N}^*$ be a repaired network obtained via* P. *The repair operation is* minimal with respect to C *iff: (i) for all $(\mathbf{x}, l) \in \mathcal{R}$ it is the case that $\mathcal{N}^*(\mathbf{x}) = l$ and (ii) $d(\mathcal{N}, \mathcal{N}^*, C) \leq d(\mathcal{N}, \mathcal{N}', C)$ for all possible networks $\mathcal{N}'$ obtained via any repair operation defined as per Definition 1.*

Although Definition 7 explicitly mentions C, it still allows for violations, i.e., some decisions for inputs in C may actually be altered during the repair process. This is intentional, as in practice there may exist cases where repairing an input in $\mathcal{R}$ will inevitably affect inputs in C (see Theorem 1).

4 REPAIR USING INFLUENCE

Since sound repair operations may not exist in general, practical approaches to repair need to pursue a balance between altering the response to inputs in $\mathcal{R}$ and affecting decisions for inputs in C. In this section we present influence-repair, or I-repair, a repair procedure that achieves such a balance.

Given an input $\mathbf{x} \in \mathcal{S}$ and a NN $\mathcal{N}$, I-repair heuristically estimates the extent to which each parameter in $\mathcal{N}$ influences the output $\mathcal{N}(\mathbf{x})$. Informally, a parameter is influential if changing its value results in a significant change in $\mathcal{N}(\mathbf{x})$. If a parameter has a

Algorithm 1 Influence estimation

1: **function** ESTIMATE_INFLUENCE($\mathcal{N}, \mathcal{X}, \psi, \mathcal{L}$)
2: $\mathcal{I} \leftarrow$ zero_array($\mathcal{N}$.num_params)
3: **for input** $(\mathbf{x}, l)$ **in** $(\mathcal{X}, \psi)$ **do**
4: $\mathcal{I}$ += $\nabla \mathcal{L}(\mathcal{N}, \mathbf{x}, l)$
5: **return** $\mathcal{I}$/len($\mathcal{X}$)

Algorithm 2 Main repair routine

1: **function** I-REPAIR($\mathcal{N}, \mathcal{X}_\mathcal{R}, \psi_\mathcal{R}, \mathcal{X}_C, \psi_C, \mathcal{L}, \gamma, ub$)
2: $\mathcal{N}^* \leftarrow \mathcal{N}$
3: $\mathcal{X}^\top_\mathcal{R}, \mathcal{X}^\bot_\mathcal{R} \leftarrow \emptyset, \emptyset$
4: **for** i = 0...ub **do**
5: $\mathcal{X}^\bot_\mathcal{R} \leftarrow \mathcal{X}_\mathcal{R}$[where $\mathcal{N}(\mathcal{X}_\mathcal{R}) \neq \psi_r$]
6: **if** $\mathcal{X}^\bot_\mathcal{R}$ **is** empty **then break**
7: $\mathcal{I}_C \leftarrow$ ESTIMATE_INFLUENCE($\mathcal{N}, \mathcal{X}_C, \psi_C, \mathcal{L}$)
8: $\mathcal{I}_{\mathcal{R}^\bot} \leftarrow$ ESTIMATE_INFLUENCE($\mathcal{N}, \mathcal{X}^\bot_\mathcal{R}, \psi_{\mathcal{R}^\bot}, \mathcal{L}$)
9: scores = $\frac{1}{\gamma} \cdot (\frac{|\mathcal{I}_{\mathcal{R}^\bot}|+\varepsilon}{|\mathcal{I}_C|+\varepsilon})$
10: **for layer** ℓ **in** $\mathcal{N}$ **do**
11: $m_\ell \leftarrow$ zero_array($\mathcal{N}$.num_params)
12: **for parameter** p **in layer** ℓ **do**
13: $m_\ell[p] \leftarrow \mathbb{1}$(scores$[p] \geq 1$)
14: ℓ.gradients $*$=m_ℓ
15: $\mathcal{N}^* \leftarrow$ BACK-PROP($\mathcal{N}, \mathcal{X}^\bot_\mathcal{R}, \psi_{\mathcal{R}^\bot}, \mathcal{L}$)
16: $\mathcal{X}^\bot_\mathcal{R} \leftarrow \mathcal{X}_\mathcal{R}$[where $\mathcal{N}(\mathcal{X}_\mathcal{R}) \neq \psi_r$]
17: $\mathcal{X}^\top_\mathcal{R} \leftarrow \mathcal{X}_\mathcal{R}$[where $\mathcal{N}(\mathcal{X}_\mathcal{R}) = \psi_r$]
18: **return** $\mathcal{N}^*, \mathcal{X}^\bot_\mathcal{R}, \mathcal{X}^\top_\mathcal{R}$

strong influence on inputs in $\mathcal{R}$, but not on inputs in C, then this parameter is a good candidate for repair, as modifying its value is unlikely to affect decisions in the correct set. Once a set of parameters is selected for repair, I-REPAIR performs back-propagation on them with the aim to fix misclassifications in $\mathcal{R}$ while keeping outputs unchanged for inputs in C; this process is repeated until all inputs are repaired or a timeout is reached.

Estimating influence with gradients. To estimate the effect of altering a parameter, we introduce Algorithm 1, a procedure that analyses gradients of each parameter in the network to repair them on the basis of their influence. The algorithm takes as input a network $\mathcal{N}$, a set of inputs $\mathcal{X}$ with corresponding (correct) labels ψ, and a loss function $\mathcal{L}$. After initialising the data structure $\mathcal{I}$ used to store influence estimates (l. 2), the algorithm iteratively computes gradients of network parameters with respect to input pairs $(\mathbf{x}, l)$ and loss $\mathcal{L}$ (l. 1). Intuitively, if the gradient of a parameter p is large, then changing the value of p will likely have significant impact on $\mathcal{L}(\mathcal{N}, \mathbf{x}, l)$. Different losses can be used to compute influence estimations, also depending on the properties of the input sets being analysed, as we will show experimentally in Section 5. Once influence has been computed for all inputs, we return an average value over the total number of inputs evaluated (l. 5). These values provide an estimate of how influential each parameter is with respect to the input set and loss considered.

The main repair procedure. We now show how the heuristic presented above is used to guide the repair operations performed by I-REPAIR. The main steps of the procedure are formalised in Algorithm 2. I-REPAIR takes as input a NN $\mathcal{N}$, sets $\mathcal{X}_\mathcal{R}$ and $\mathcal{X}_C$ with their respective (correct) labels $\psi_\mathcal{R}, \psi_C$, a loss function $\mathcal{L}$, a real number $\gamma \geq 0$ and an integer $ub \geq 0$.

After performing some initialisation, the algorithm enters the main loop where influence estimation is performed on both correct and repair sets (l. 7–8). Notice that at each iteration, the repair set is updated to contain only those inputs that remain misclassified after each repair iteration; if no inputs are left in the repair set, the main loop is halted (l. 6).

Since the algorithm aims to locate repair candidates that have a large influence on the repair set ($\mathcal{I}_{\mathcal{R}^\bot}$) and small influence on the correct set ($\mathcal{I}_C$), parameters in $\mathcal{N}$ are assigned a score according to their ratio (l. 9).[1] The multiplicative coefficient γ weighs the relative importance of $\mathcal{I}_{\mathcal{R}^\bot}$ and $\mathcal{I}_C$ and is used to allow more flexibility in the repair process.

After scores have been assigned, the procedure traverses $\mathcal{N}$ layer by layer and freezes gradients of those parameters whose score is below one (l. 10–14). Intuitively, if a parameter p has score greater than one, then the average influence of p on $\mathcal{R}$ is higher than that on C, i.e., changing p will likely affect erroneous predictions more than correct ones. Standard back-propagation is then performed to update network parameters that were not frozen in the previous step.

The main loop repeats the steps presented above until either all inputs are repaired, or a user-defined upper bound ub is reached. Notice that this upper bound is needed to ensure termination as it prevents I-REPAIR from entering infinite loops when an input cannot be repaired (see Theorem 1). Following this, we report the inputs that could be repaired and those that could not be before terminating (l. 16–17).

OBSERVATION 1. *Consider a NN $\mathcal{N}$, a correct set C and a repair set $\mathcal{R}$. Let $\mathcal{N}^*$ be the repaired network returned by* I-REPAIR*, together with the two sets $\mathcal{X}^\top_\mathcal{R}$ and $\mathcal{X}^\bot_\mathcal{R}$. Then for all $\mathbf{x} \in \mathcal{X}^\top_\mathcal{R}$ we have $\mathcal{N}^*(\mathbf{x}) = l$ where l is the true label corresponding to $\mathbf{x}$.*

That is, when I-REPAIR reports that an input has been repaired, this input is guaranteed to satisfy the desired specification. This property directly follows from the fact that I-REPAIR terminates only after checking that $\mathcal{N}^*$ correctly classifies repair inputs in $\mathcal{X}^\top_\mathcal{R}$(l. 17, Algorithm 2).

OBSERVATION 2. I-REPAIR *converges to standard back-propagation when $\gamma \to 0$.*

This observation highlights that I-REPAIR inherits optimality guarantees of the optimisation algorithm used for back-propagation. If stochastic gradient descent is used, I-REPAIR converges to globally optimal solutions for $\gamma \to 0$ when $\mathcal{L}$ is convex wrt the parameters of $\mathcal{N}$ [6].

Observation 2 provides sufficient conditions for I-REPAIR to operate optimal repair operations, thereby showing the potential of the algorithm. These conditions may or may not hold in practical examples. However, we show experimentally in the next section that the algorithm produces repaired networks of good characteristics in a scalable manner.

[1] An arbitrarily small ε is added to avoid division by zero.

5 EVALUATION

Section 4 laid the foundations for a novel repair procedure that is able to fix misclassifications using only limited data. We now present an efficient implementation and evaluate its performance on a wide range of experiments. In order to validate our approach, we consider the following research questions (RQs): *(i)* is I-REPAIR successful in repairing misclassifications when only limited data is available? *(ii)* is the influence heuristic informative? *(iii)* does I-REPAIR cause overfitting? *(iv)* is our minimality criterion effective?

Comparison with other approaches. As discussed in Section 1, most repair approaches assume access to the full training data and are thus not suitable for a direct comparison. While approaches such as [30, 35] could have been adapted to the limited data setting, their source code was not released. However, we were able to recreate the minimal modifications (MM) repair approach from [13] with help from the authors. MM repair is a fully data-free approach that tries to find minimal modifications to the weights of the last layer of the NN such that misclassification are corrected. To find such optimal modifications, MM repair performs a binary search in an ε-neighbourhood of the original weights.

We implemented Algorithms 1 and 2 in a Python toolkit called I-REPAIR – see supplementary material. I-REPAIR utilises PyTorch for an efficient GPU-based implementation of the influence estimation and back-propagation. We compared the performance of I-REPAIR against MM repair as well as standard retraining. Retraining consists in running a new training phase on the available data only. We used the RMSprop optimiser [21] for both retraining and I-REPAIR. Also note that retraining on the full training data is not a viable option here, as our working assumption is that such data is not available; thus, we did not include this approach in our experimental analysis.

Setup. We use two popular architectures trained on commonly used datasets: a ResNet model [16] composed of 9 blocks, each containing 2 convolutional layers followed by a batch normalisation layer, and a fully connected network with 2 hidden layers and 256 ReLU nodes each. The ResNet network was trained on the CIFAR10 dataset [27], while the fully connected network was trained on the MNIST [29] dataset; test-set accuracies are 90.22% and 97.36%, respectively.

Each dataset was split into four parts of 45k, 5k, 5k and 5k data points. The first two splits were used for training and validation during the training phase; the third split was used to extract the repair and correct set, simulating a scenario where repair does not have access to the original training data. The last split was used to calculate accuracies after repair and evaluate possible overfitting caused by repair.

Training and repair were performed on a machine with a Ryzen 3700X 3.6 GHz 8-core CPU, Nvidia RTX 2080 GPU, 64 GB ram and Ubuntu 20.04 with Linux kernel 5.4.0.

RQs *(i)*-*(iii)* are evaluated using the CIFAR10 ResNet as a benchmark. To answer RQ *(iv)* a comparison against MM repair is required; the approach however does not support ResNet architectures. We therefore use the fully-connected MNIST model to answer this RQ.

RQ *(i)*. For this RQ we seek to investigate whether I-REPAIR is successful in correcting misclassifications when only limited data is available. We consider two scenarios: first we study repair problems where all misclassification occur in the same class; we then move to a scenario where misclassifications are spread over several classes.

Scenario 1. For these experiments we used repair sets of size 1, 2, 4, 8, 16 and 32 for each output class and correct sets of size 40 with inputs uniformly distributed among all classes. For each of the 10 CIFAR classes, we ran I-REPAIR and retraining[2] and recorded the accuracy drop observed for each class and repair set size. To evaluate the statistical significance this evaluation was repeated 20 times with different random seeds. We ran I-REPAIR and retraining for a maximum of 5000 iterations or until all inputs were repaired. Some classes had less than 32 misclassified inputs: in such cases we used all misclassified inputs available for a total of 303 repair inputs.

Figure 1a reports the accuracy drop averaged over all classes as well as the 95%-confidence intervals (repair set sizes are shown on the x-axis). Both I-REPAIR and retraining succeeded in repairing all misclassification across all repair sets considered. However, an evaluation of the post-repair accuracy reveals that repair operations performed by I-REPAIR resulted in repaired networks with better performance. To confirm this, we calculated the p-values for each repair set using the independent two-sample t-test and obtained the following values $0.149, 0.019, 0.001, < 0.001, \ll 0.001, \ll 0.001$, thus confirming the advantages of I-REPAIR with statistical significance ($p < 0.05$) for repair sets larger than 1. We note that I-REPAIR is computationally more demanding than retraining as it requires calculating influence scores and normally uses more iterations; however the runtime was not prohibitive, e.g., with 8 repair inputs I-REPAIR used 3017s for one run with all classes while retraining used 472s.

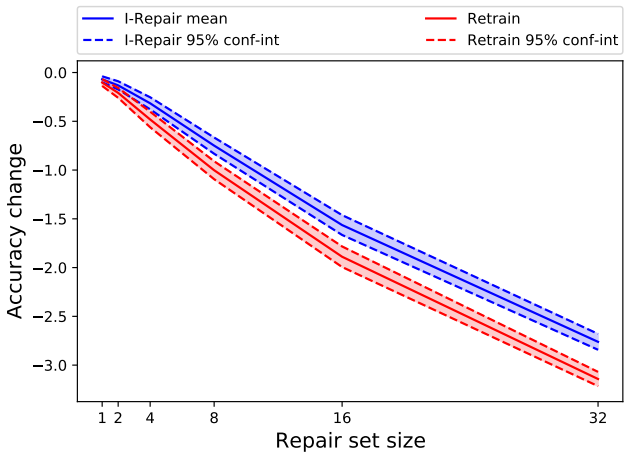
Scenario 2. We now analyse the performance of I-REPAIR on repair sets containing different classes. The repair sets were created by extracting repair inputs from one to ten different classes. For each class we used 1, 2 and 4 repair inputs and averaged the accuracy change. Results for $\gamma = 2$ and correct sets of size 40 are presented in Figure 1d. Again, our results show that I-REPAIR outperforms retraining also for multi-class repairs, with the relative gap in performance being larger when fewer classes are considered.

Answer to RQ (i). I-REPAIR shows benefits in all the scenarios considered. It can successfully repair a NN using only limited data. Retraining also succeeds, but I-REPAIR results in a statistically significant ($p < 0.05$) higher accuracy for repair sets larger than 1.

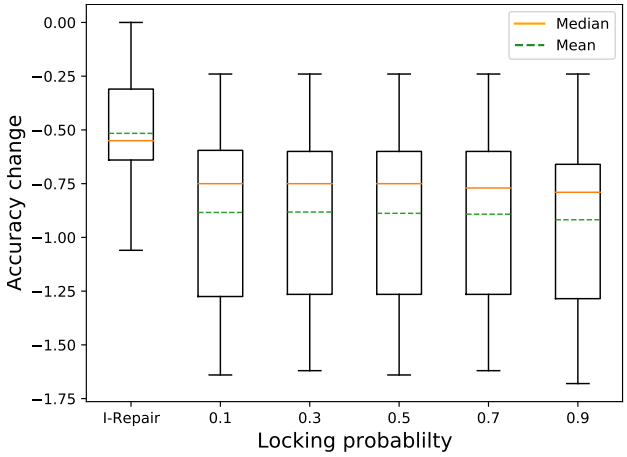
RQ *(ii)*. We now explore the impact that the influence-based heuristic has on I-REPAIR. To evaluate whether our heuristic is informative we ran two sets of experiments reported below.

Randomised Repair. We compare I-REPAIR against a version that locks random parameters on the ResNet model. Given a probability threshold p, a random number is drawn in $[0, 1]$ for each trainable parameter in the network. Parameters for which this number is below p are frozen and cannot be modified by repair, while remaining parameters are updated as usual by back propagation.

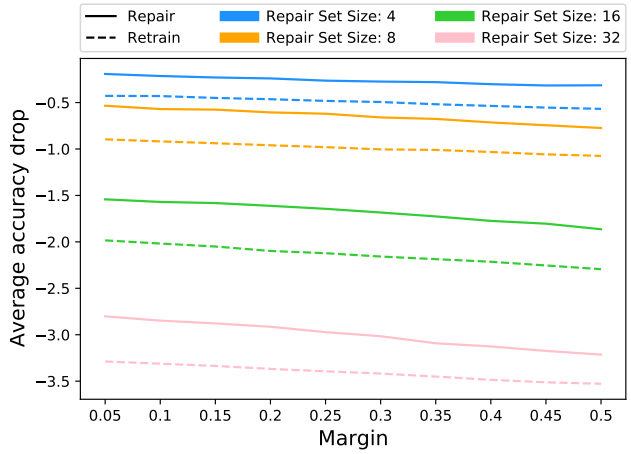
[2]We experimentally determined $\gamma = 2$ to work well (see supplementary material) and compared I-REPAIR to retraining for all repair set sizes. The back-propagation phase used a learning rate of 10^{-5} for both I-REPAIR and retraining.



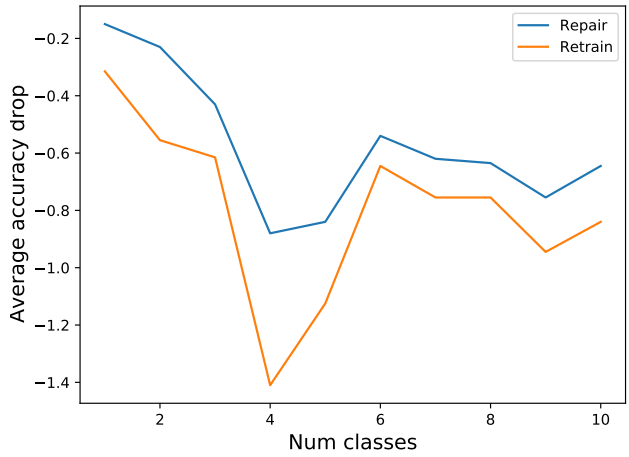
(a) Repair vs Retraining.



(b) Random vs influence-based locking.



(c) Varying stopping conditions.



(d) Multi-class repair.

Figure 1: ResNet: (a) accuracy drop after repair and retraining for CIFAR10 networks with confidence intervals, (b) repair with influence-based locking vs random locking, (c) evaluation of different termination conditions for I-REPAIR and (d) accuracy drop for multi-class repair.

We used repair sets of size 8, correct sets of size 40 and $\gamma = 2$ in these experiments. Results are presented in Figure 1b, with different probabilities shown on the x-axis.

The influence-based heuristic consistently outperforms the randomised approach, achieving almost 50% less accuracy drop. Note that the accuracy change induced by random locking is comparable across different locking probabilities, with the average accuracy change decreasing for increased probabilities (-0.884, -0.882, -0.888, -0.892 and -0.918 for probabilities of 0.1, 0.3, 0.5, 0.7 and 0.9, respectively). We hypothesise that the network has enough capacity to overfit to the repair and correct sets used, even when a large amount of parameters are randomly locked. In contrast, targeted parameter-locking via influence estimation seems to reduce overfitting in I-REPAIR.

Influence loss-function. As mentioned in the previous section, different losses can be used to calculate influence in Algorithm 1. In the following we compare the cross-entropy loss against the sum of the output magnitudes, i.e., $\mathcal{L}(\mathcal{N}, \mathbf{x}) = \sum_i |\mathcal{N}(\mathbf{x})_i|$ for the influence of the correct set. While cross-entropy is commonly used in machine learning, the same does not hold for absolute magnitude loss. However, we argue that the latter is more appropriate when evaluating influence on correct inputs as the gradients of this loss are large for all parameters that significantly affect the output thus encouraging the locking of these parameters. This is exactly our requirement for the correct set as we aim to keep the network behaviour unchanged for correct inputs.

To confirm our hypothesis, we ran tests with $\gamma = 2$, correct set of size 40 and repair sets of size 1, 2, 4 and 8. The runs that used cross-entropy to evaluate influence on both sets had an average accuracy decrease twice as large compared to the runs that used cross-entropy for the repair set and absolute magnitude for the correct set (-0.2 vs -0.4).

Answer to RQ (ii). The results above confirm that our heuristic in indeed informative and can thus be used to guide meaningful repair interventions. Our experiments also indicate that the absolute magnitude loss better captures the notion of minimality as formulated in Definition 7.

RQ *(iii).* We now investigate whether the operations performed by I-REPAIR may result in overfitted NNs.

Following standard practice, we used a test-set to calculate the network accuracy in all the experiments above. This evaluation takes into account traditional notions of overfitting, which our approach mitigates compared to retraining. However, due to the stopping condition used in Algorithm 2, l. 6 and retraining, repaired misclassifications may lie close to the decision boundary and thus be fragile.

To alleviate the above problem, the stopping criterion can be extended such that an input is considered to be repaired if the output score for the correct class c exceeds the scores for all other classes by a user-defined margin, i.e., $\mathcal{N}(\mathbf{x})_c - \mathcal{N}(\mathbf{x})_i > \delta$ for all $i \neq c$. The ResNet we use here applies Softmax to its output thus δ is always in the range $[0, 1]$.

Figure 1c shows the results obtained for I-REPAIR and retraining operations for several margins, general set of size 40 and repair sets of sizes $\{4, 8, 16, 32\}$. As we can observe, I-REPAIR consistently outperforms retraining also under the new termination condition. Overall, smaller margins result in lower drops in accuracy for both approaches. This is to be expected as larger margins may increase the magnitude of parameter changes that are required for repair.

Answer to RQ (iii). Our results confirms that I-REPAIR reduces overfitting compared to retraining. Resulting repairs may be fragile; however, they can be strengthened by extending the stopping conditions.

RQ *(iv).* We now evaluate the effectiveness of our approach in producing repair operations that do not significantly reduce the overall network accuracy. We compare against retraining and MM repair [13]. In contrast to the minimality criterion used by I-REPAIR, MM repair minimises weight changes as a way to produce less invasive repair operations. We considered single-class repair problems with repair sets of size 1, 2 and 4 and learning rate of 10^{-4} in both repair and retraining. All other hyperparameters were the same as in the CIFAR10 experiments. MM repair was run using default parameters and $\varepsilon \in \{1, 5\}$.

Table 1 presents the accuracy drop after repair and retraining averaged over all 10 classes. We observed significant gains when comparing I-REPAIR to both retraining and MM repair. For instance, when $\gamma = 1$ I-REPAIR successfully repairs all inputs with only ∼13% of the accuracy drop caused by both MM repair and retraining. The performance of retraining and MM repair are comparable in terms of accuracy drop. I-REPAIR and retraining achieved comparable running times, while MM repair took considerably more, e.g., repairing all the 10 classes with $|\mathcal{R}| = 4$ took I-REPAIR 400s ($\gamma = 1$), as opposed to 340s for retraining and 6840s ($\varepsilon = 1$, 1 class failed) and 8400s ($\varepsilon = 5$) for MM repair. We also highlight that MM repair may fail depending on the value of ε as observed in Table 1, where repair fails for one class when $\varepsilon = 1$. Consequently one must be careful when choosing ε, as small values may lead to failures in repair while a large values may result in longer running times. In contrast, I-REPAIR and retraining succeeded in repairing all classes.

Table 1: MNIST experiments using repair sets of size 1,2 and 4. d indicates the accuracy drop obtained for single class repair, averaged over the ten classes considered; s is the standard deviation. * One instance failed as ε was too small.

	$\lvert\mathcal{R}\rvert = 1$		$\lvert\mathcal{R}\rvert = 2$		$\lvert\mathcal{R}\rvert = 4$	
	d	s	d	s	d	s
Retrain	1.51	1.47	1.42	1.33	1.40	0.99
I-REPAIR, $\gamma = 0.01$	0.87	1.47	0.99	1.63	0.37	0.40
I-REPAIR, $\gamma = 0.05$	0.62	1.16	0.88	1.32	0.34	0.35
I-REPAIR, $\gamma = 0.1$	0.54	0.94	0.80	1.17	0.33	0.34
I-REPAIR, $\gamma = 0.5$	0.41	0.71	0.18	0.19	0.21	0.25
I-REPAIR, $\gamma = 1$	0.13	0.24	0.15	0.17	0.18	0.21
MM Repair, $\varepsilon = 1$	1.56	3.01	1.41	2.60	1.44*	2.06*
MM Repair, $\varepsilon = 5$	1.56	3.01	1.41	2.60	1.30	1.99

Table 1 reports results for repair sets of sizes up to a maximum of 4 as for some classes we could not find enough misclassified images, e.g., only up to 4 misclassifications could be found for class 0. However, we did run further experiments with repair sets of size 8 for those classes that allowed. For these larger sets, MM repair always timed out after 1 hour, while I-REPAIR generally showed accuracy drops comparable to retraining. We postulate that these failures are due to properties of MNIST, as CIFAR10 experiments showed a rather different picture with I-REPAIR consistently outperforming retraining, even for larger repair sets.

Answer to RQ (iv). The experimental evaluation reveals that our proposed notion of minimality does indeed result in repaired networks that have higher accuracy than the weight minimality used in MM repair.

6 CONCLUSIONS

We studied the problem of repairing a set of misclassifications in a NN without access to the training set. We analysed theoretical limitations of the problem and identified new criteria to guide repair interventions. We incorporated these results into I-REPAIR, a novel algorithm that achieves scalable repair while heuristically minimising the effect that repair has on the output of a network. Our experiments demonstrated that I-REPAIR can successfully repair misclassifications using only limited data. Our analysis confirmed that approach is able to reduce accuracy drops by up to 90% compared to the state of the art.

NN repair is still in its infancy and several avenues of research remain to be explored. In future work, we plan to investigate techniques to make repair even less invasive and look into methods for combining formal verification methods with repair to improve robustness in NNs.

ACKNOWLEDGEMENTS

This work was supported by the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (EP/S023356/1), the

DARPA Assured Autonomy programme (FA8750-18-C-0095), and the UK Royal Academy of Engineering (CiET17/18-26).

REFERENCES

[1] M. Akintunde, E. Botoeva, P. Kouvaros, and A. Lomuscio. 2020. Verifying Strategic Abilities of Neural-symbolic Multi-agent Systems. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR20)*. IJCAI Press, 22–32.

[2] M.E. Akintunde, E. Botoeva, P. Kouvaros, and A. Lomuscio. 2022. Formal verification of neural agents in non-deterministic environments. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 1–36.

[3] S. Bak, H. Tran, K. Hobbs, and T. Johnson. 2020. Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification (CAV20) (LNCS)*, Vol. 12224. Springer, 66–96.

[4] B. Battern, P. Kouvaros, A. Lomuscio, and Y. Zheng. 2021. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In *In proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*. IJCAI, 2184–2190.

[5] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. 2020. Efficient Verification of Neural Networks via Dependency Analysis. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI20)*. AAAI Press.

[6] L. Bottou. 2003. Stochastic Learning. In *Advanced Lectures on Machine Learning (LNCS)*, Vol. 3176. Springer, 146–168.

[7] R. Bunel, J. Lu, I. Turkaslan, P.H.S. Torr, P. Kohli, and M.P. Kumar. 2020. Branch and Bound for Piecewise Linear Neural Network Verification. *Journal of Machine Learning Research (JMLR20)* 21, 42 (2020), 1–39.

[8] C.H Cheng. 2021. Provably-Robust Runtime Monitoring of Neuron Activation Patterns. In *Design, Automation & Test in Europe Conference & Exhibition (DATE21)*. IEEE, 1310–1313.

[9] C.H. Cheng, G. Nührenberg, and H. Yasuoka. 2019. Runtime Monitoring Neuron Activation Patterns. In *Design, Automation & Test in Europe Conference & Exhibition (DATE19)*. IEEE, 300–303.

[10] C-H. Cheng and R. Yan. 2021. Continuous Safety Verification of Neural Networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE21)*. 1478–1483.

[11] A. De Palma, R. Bunel, A. Desmaison, K. Dvijotham, P. Kohli, P. Torr, and M. Kumar. 2021. Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition. *arXiv preprint 2104.06718* (2021).

[12] T. Dreossi, S. Ghosh, X. Yue, K. Keutzer, A. Sangiovanni-Vincentelli, and S. Seshia. 2018. Counterexample-guided data augmentation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence and 23rd European Conference on Artificial Intelligence (IJCAI-ECAI18)*. AAAI press, 2071–2078.

[13] B. Goldberger, G. Katz, Y. Adi, and J. Keshet. 2020. Minimal Modifications of Deep Neural Networks using Verification. In *Proceedings of the 23th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR20)*, Vol. 73. EasyChair, 260–278.

[14] A. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep learning*. MIT press Cambridge.

[15] D. Guidotti, F. Leofante, C. Castellini, and A. Tacchella. 2019. Repairing Learned Controllers with Convex Optimization: A Case Study. In *Proceedings of the 16th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR19)*, Vol. 11494. Springer, 364–373.

[16] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR16)*. IEEE Computer Society, 770–778.

[17] P. Henriksen, K. Hammernik, D. Rueckert, and A. Lomuscio. 2021. Bias Field Robustness Verification of Large Neural Image Classifiers. In *In proceedings of the 32nd British Machine Vision Conference (BMVC21)*.

[18] P. Henriksen and A. Lomuscio. 2020. Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI20)*. IOS Press, 2513–2520.

[19] P. Henriksen and A. Lomuscio. 2021. DEEPSPLIT: An Efficient Splitting Method for Neural Network Verification via Indirect Effect Analysis. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization, 2549–2555.

[20] T.A. Henzinger, A. Lukina, and C. Schilling. 2020. Outside the Box: Abstraction-Based Monitoring of Neural Networks. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI20)*. IOS Press.

[21] G. Hinton. 2012. Lecture 6E. RMSprop: Divide the gradient by a running average of its recent magnitude. http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf.

[22] S. Ioffe and C. Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32th International Conference on Machine Learning (ICML15)*. JMLR.org, 448–456.

[23] G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. Dill, M. Kochenderfer, and C. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proceedings of the 31th International Conference on Computer Aided Verification (CAV19) (LNCS)*, Vol. 11561. Springer, 443–452.

[24] P. Kouvaros, T. Kyono, F. Leofante, A. Lomuscio, D. Margineantu, D. Osipychev, and Y. Zheng. 2021. Formal Analysis of Neural Network-Based Systems in the Aircraft Domain. In *International Symposium on Formal Methods (Lecture Notes in Computer Science)*, Vol. 13047. Springer, 730–740.

[25] P. Kouvaros and A. Lomuscio. 2018. Formal Verification of CNN-based Perception Systems. *arXiv preprint arXiv:1811.11373* (2018).

[26] P. Kouvaros and A. Lomuscio. 2021. Towards scalable complete verification of relu neural networks via dependency-based branching. In *In proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*. IJCAI, 2643–2650.

[27] A. Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report.

[28] J. Lan, A. Lomuscio, and Y. Zheng. 2021. Tight Neural Network Verification via Semidefinite Relaxations and Linear Reformulations.. In *Proccedings of the 36th AAAI Conference on Artificial Intelligence (AAAI22)*. AAAI Press, To appear.

[29] Y. Lecun. 1999. THE MNIST DATABASE of handwritten digits. https://ci.nii.ac.jp/naid/10027939599/en/.

[30] S. Ma, Y. Liu, W. Lee, X. Zhang, and A. Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 ACM Symposium on the Foundations of Software Engineering FSE18*. ACM, 175–186.

[31] L. Pulina and A. Tacchella. 2010. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV10) (LNCS)*, Vol. 6184. Springer, 243–257.

[32] M. Ribeiro, K. Grolinger, and M. Capretz. 2015. MLaaS: Machine Learning as a Service. In *Proceedings of the 14th IEEE International Conference on Machine Learning and Applications (ICMLA'15)*. IEEE, 896–902.

[33] T. Serra, C. Tjandraatmadja, and S. Ramalingam. 2018. Bounding and Counting Linear Regions of Deep Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML18) (PMLR)*, Vol. 80. PMLR, 4565–4573.

[34] G. Singh, T. Gehr, M. Püschel, and M. Vechev. 2019. An abstract domain for certifying neural networks. *In proceedings of the ACM on Programming Languages* 3, POPL (2019), 41.

[35] J. Sohn, S. Kang, and S. Yoo. 2019. Search Based Repair of Deep Neural Networks. *arXiv preprint 1912.12463* (2019).

[36] M. Sotoudeh and A. Thakur. 2019. Correcting Deep Neural Networks with Small, Generalizing Patches. In *Workshop on Safety and Robustness in Decision Making*.

[37] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore. 2019. DeepConcolic: testing and debugging deep neural networks. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 111–114.

[38] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore. 2019. Structural test coverage criteria for deep neural networks. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 320–321.

[39] H. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. Nguyen, W. Xiang, S. Bak, and T. Johnson. 2020. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In *Proceedings of the 32rd International Conference on Computer Aided Verification (CAV20) (LNCS)*, Vol. 12224. Springer, 3–17.

[40] M. Usman, D. Gopinath, Y. Sun, Y. Noller, and C. Pasareanu. 2021. NNrepair: Constraint-Based Repair of Neural Network Classifiers. In *Proceedings of the 33rd International Conference on Computer Aided Verification (CAV21) (LNCS)*, Vol. 12759. Springer, 3–25.

[41] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. 2018. Efficient Formal Safety Analysis of Neural Networks. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems 2018 (NeurIPS2018)*. Curran Associates, Inc., 6367–6377.

[42] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. 2018. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proceedings of the 27th USENIX Security Symposium (USENIX18)*.

[43] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C. Hsieh, , and Z. Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Verification. *arXiv preprint 2103.06624* (2021).

[44] H. Zhang and W. K. Chan. 2019. Apricot: A Weight-Adaptation Approach to Fixing Deep Learning Models. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE19)*. IEEE, 376–387.