QNNRepair: Quantized Neural Network Repair

Xidan Song[1][0000−0003−2612−6296], Youcheng Sun[1][0000−0002−1893−6259], Mustafa A. Mustafa[1,2][0000−0002−8772−8023], and Lucas C. Cordeiro[1,3][0000−0002−6235−4272]

[1] Department of Computer Science, The University of Manchester, UK
[2] COSIC, KU Leuven, Belgium
[3] Federal University of Amazonas, Brazil
{xidan.song, youcheng.sun, mustafa.mustafa, lucas.cordeiro}@manchester.ac.uk

Abstract. We present QNNRepair, the first method in the literature for repairing quantized neural networks (QNNs). QNNRepair aims to improve the accuracy of a neural network model after quantization. It accepts the full-precision and weight-quantized neural networks, together with a repair dataset of passing and failing tests. At first, QNNRepair applies a software fault localization method to identify the neurons that cause performance degradation during neural network quantization. Then, it formulates the repair problem into a linear programming problem of solving neuron weights parameters, which corrects the QNN's performance on failing tests while not compromising its performance on passing tests. We evaluate QNNRepair with widely used neural network architectures such as MobileNetV2, ResNet, and VGGNet on popular datasets, including high-resolution images. We also compare QNNRepair with the state-of-the-art data-free quantization method SQuant [21]. According to the experiment results, we conclude that QNNRepair is effective in improving the quantized model's performance in most cases. Its repaired models have 24% higher accuracy than SQuant's in the independent validation set especially for the ImageNet dataset.

Keywords: neural network repair, quantization, fault localization, constraints solving

1 Introduction

Nowadays, neural networks are often used in safety-critical applications, such as autonomous driving, medical diagnosis, and aerospace systems [51]. In such applications, often quantized (instead of full precision) neural network models are deployed due to the limited computational and memory resources of embedded devices [22]. Since the consequences of a malfunction/error in such applications can be catastrophic, it is crucial to ensure that the network behaves correctly and reliably [40].

Quantized neural networks [22] use low-precision data types, such as 8-bit integers, to represent the weights and activations of the network. While this reduces the memory and computation requirements of the network, it can also lead to a loss of accuracy and the introduction of errors in the network's output. Therefore, it is important to verify that the quantization process has not introduced any significant errors that could affect the safety or reliability of the network.

To limit the inaccuracy in a specific range, various neural network model verification methods [14] [29] [28] [46] [24] have been proposed. Neural network verification [25,38,41] aims to provide formal guarantees about the behavior of a neural network, ensuring that it meets specific safety and performance requirements under all possible input conditions. They set constraints and properties of the network input and output to check whether the model satisfies the safety properties. However, neural network verification can be computationally expensive, especially for large, deep networks with millions of parameters. This can make it challenging to scale the verification process to more complex models. While the majority of the neural network verification work is on full precision models, many verification techniques focus on quantized models as well [24,50,18,4].

Other researchers improve the performance and robustness of the trained neural network models by repair [49] [19] [44] [43] [6]. These methods can be divided into three categories: retraining/refining, direct weight modification, and attaching repairing units. There are also quantized aware training (QAT) techniques [31] [20] [8], a method to train neural networks with lower precision weights and activations, typically INT8 format. QAT emulates the effects of quantization during the training process. QAT requires additional steps, such as quantization-aware back-propagation and quantization-aware weight initialization, making the training process more complex and time-consuming. Quantized-aware training methods require datasets for retraining, which consume a lot of time and storage. However, for Data-free quantization like SQuant [21], which does not require datasets, the accuracy after quantization is relatively low.

In QNNRepair, we use the well-established software fault localization methods to identify suspicious neurons in a quantized model corresponding to the performance degradation after quantization. We then correct these most suspicious neurons' behavior by linear programming, in which the constraints are encoded by observing the difference between the quantized model and the original model when inputs are the same. The main contributions of this paper are three-fold:

- We propose, implement and evaluate QNNRepair – a new method for repairing QNNs. It converts quantized neural network repair into an LP (linear programming) problem. QNNRepair features direct weight modification and does not require the training dataset.
- We compare QNNRepair with a state-of-the-art QNN repair method – Squant [21], and demonstrate that QNNRepair can achieve higher accuracy than Squant after repair. We also evaluate QNNRepair on multiple widely used neural network architectures to demonstrate its effectiveness.
- We have made QNNRepair and its benchmark publicly available at: `https://github.com/HymnOfLight/QNNRepair`

2 Related Work

2.1 Neural Network Verification

The first applicable methods supporting the non-linear activation function for neural network verification can be traced back to 2017, R Ehlers et al. [14] proposed the first practicable neural network verification method based on SAT solver (solve the Boolean satisfiability problem) [13]. They present an approach to verify neural networks with piece-wise linear activation functions. Guy Katz et al. [29] presented Marabou, an SMT(Satisfiability modulo theories) [11]-based tool that can answer queries about a network's properties by transforming these queries into constraint satisfaction problems. However, implementing SMT-based neural network verification tools is limited due to the search space and the scale of a large neural network model, which usually contains millions of parameters [17]. The SMT-based neural network verification has also been proved as NP-complete problem [28]. Shiqi Wang et al. develop β-CROWN [46], a new bound propagation-based method that can fully encode neuron splits via optimizable parameters β constructed from either primal or dual space. Their algorithm is empowered by the α,β-CROWN (alpha-beta-CROWN) verifier, the winning tool in VNN-COMP 2021 [5]. There are also some quantized neural network verification methods. TA Henzinger et al. [24] proposed a scalable quantized neural network verification method based on abstract interpretation. But due to the search-space explosion and it has been proved that SMT-based quantized neural network verification is a PSPACE-hard problem [24].

2.2 Neural Network Repair

Many researchers have proposed their full-precision neural network repairing techniques. These can be divided into three categories: Retraining, direct weight modification, and attaching repairing units.

In the first category of repair methods, the idea is to retrain or fine-tune the model for the corrected output with the identified misclassified input. DeepRepair [49] implements transfer-based data augmentation to enlarge the training dataset before fine-tuning the models. The second category use solvers to get the corrected weights and modify the weight in the trained model directly. These types of methods, including [19] and [44], used SMT solvers for solving the weight modification needed at the output layer for the neural network to meet specific requirements without any retraining. The third category of methods repairs the models by introducing more weight parameters or repair units to facilitate more efficient repair. PRDNN [43] introduces a new DNN architecture that enables efficient and effective repair, while DeepCorrect [6] corrects the worst distortion-affected filter activations by appending correction units. AIRepair [42] aims to integrate multiple existing repair techniques into the same platform. However, these methods only support the full-precision models and cannot apply to quantized models.

2.3 Quantized Aware Training

Some researchers use quantized-aware training to improve the performance of the quantized models. Yuhang Li et al. proposed a post-training quantization framework by analyzing the second-order error called BRECQ(Block Reconstruction Quantization) [31]. Ruihao Gong et al. proposed Differentiable Soft Quantization (DSQ) [20] to bridge the gap between the full-precision and low-bit networks. It can automatically evolve during training to gradually approximate the standard quantization. J Choi et al. [8]proposed a novel quantization scheme PACT(PArameterized Clipping acTivation) for activations during training - that enables neural networks to work well with ultra-low precision weights and activations without any significant accuracy degradation. However, these methods require retraining and the whole dataset, which will consume lots of computing power and time to improve tiny accuracy in actual practices. In addition, there is a method called Data-free quantization, which quantizes the neural network model without any datasets. Cong Guo et al. proposed SQuant [21], which can quantize networks on inference-only devices with low computation and memory requirements.

3 Preliminaries

3.1 Statistical Fault Localization

Statistical fault localization techniques (SFL) [34] have been widely used in software testing to aid in locating the causes of failures of programs. During the execution of each test case, data is collected indicating the executed statements. Additionally, each test case is classified as passed or failed.

This technique uses information about the program's execution traces and associated outcomes (pass/fail) to identify suspicious program statements. It calculates four suspiciousness scores for each statement based on the correlation between its execution and the observed failures. We use the notation $C^{\mathrm{af}}, C^{\mathrm{nf}}, C^{\mathrm{as}}, C^{\mathrm{ns}}$. The first part of the superscript indicates whether the statement was executed/"activated" (a) or not (n), and the second indicates whether the test is a passing/successful (s) or failing (f) one. For example, C^{as} is the number of successful tests that execute a statement C. Statements with higher suspiciousness scores are more likely to contain faults. There are many possible metrics that have been proposed in the literature. We use Tarantula [27], Ochiai [2], DStar [47], Jaccard [3], Ample [9], Euclid [16] and Wong3 [48], which are widely used and accepted in the application of Statistical fault localization, in our ranking procedure. We discuss the definition and application in our method of these metrics in Section 4.1.

3.2 Neural Network and Quantization

A neural network consists of an input layer, an output layer, and one or more intermediate layers called hidden layers. Each layer is a collection of nodes, called neurons. Each neuron is connected to other neurons by one or more directed edges [15].

Let $f : \mathcal{I} \rightarrow \mathcal{O}$ be the neural network N with m layers. In this paper, we focus on a neural network for image classification. For a given input x $\in \mathcal{I}, f(x) \in \mathcal{O}$ calculates the output of the DNN, which is the classification label of the input image. Specifically, we have

$$f(x) = f_N\left(\ldots f_2\left(f_1\left(x; W_1, b_1\right); W_2, b_2\right) \ldots ; W_N, b_N\right) \tag{1}$$

In this equation, W_i and b_i for $i = 1, 2, \ldots, N$ represent the weights and bias of the model, which are trainable parameters. $f_i\left(z_{i-1}; W_{i-1}, b_{i-1}\right)$ is the layer function that maps the output of layer $(i-1)$, i.e., z_{i-1}, to the input layer i.

Quantization As one of the general neural network model optimization methods, model quantization can reduce the size and model inference time of DNN models and their application to most models and different hardware devices. Quantization is the process of approximating continuous values of a signal to a finite number of discrete values. It can be understood as a method of information compression. When considering this concept in the context of computer systems, it is generally referred to as “low bits.” In the following formula, r is the true floating point value, q is the quantized fixed point value, Z is the quantized fixed point value corresponding to the 0 floating point value, and S is the smallest scale that can be represented after quantization of the fixed point. The formula for quantization from floating point to fixed point is as follows:

$$\begin{gathered} r = S(q - Z) \\ q = \operatorname{round}\left(\frac{r}{S} + Z\right) \end{gathered} \tag{2}$$

Currently, Google’s TensorFlow Lite [10] and NVIDIA’s TensorRT [45] support the INT8 engine framework.

3.3 Linear program solver

Linear programming (LP) is a method in the field of mathematical optimization concerned with maximizing or minimizing some linear objective function while satisfying a set of linear constraints.

The state-of-the-art linear programming solvers include Gurobi [36], which is a commercial solver widely used for linear programming, integer programming, and mixed integer linear programming. According to B. Meindl and M. Templ’s [33] Analysis of commercial and free and open source solvers for linear optimization problems, Gurobi is the fastest solver and can solve the largest number of problems. Another reason for choosing Gurobi was primarily in the area of neural network robustness, and other approaches, such as alpha-beta-crown in the area of neural network verification, use Gurobi as their backend. Hence we use Gurobi as the backend to solve the neural network repairing problem.

Other linear program solver include: CPLEX [35], GLPK (GNU Linear Programming Kit) [32]. Python external library Scipy [7] also provides some functions for linear programming.

4 QNNRepair Methodology

The overall workflow of QNNRepair is illustrated in Figure 1. It takes two neural networks, a floating-point model and its quantized version for repair, as inputs. There is also a repair dataset of successful (passing) and failing tests, signifying whether the two models would produce the same classification outcome when given the same test input.

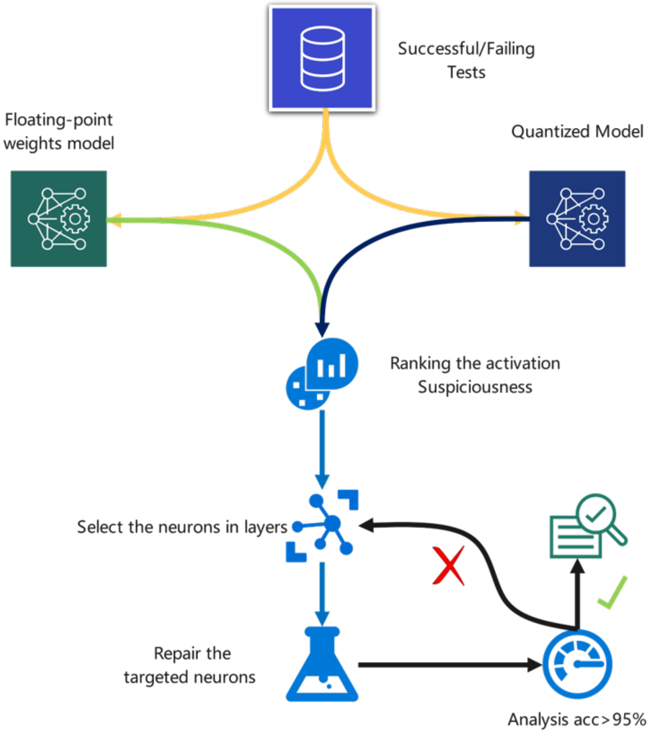
Successful/Failing Tests
Floating-point weights model
Quantized Model
Ranking the activation Suspiciousness
Select the neurons in layers
Repair the targeted neurons
Analysis acc>95%

Fig. 1: The QNNRepair Architecture.

The passing/failing tests are used by QNNRepair to evaluate each neuron's importance and localize these neurons to repair for improving the quantized model's performance (Section 4.1). In QNNRepair, the neural network repair problem is encoded into a linear programming problem for solving the corrected neuron weights (Section 4.2). It then replaces the weights with corrected weights, after which QNNRepair evaluates the performance similarity between the floating model and the quantized model on an independent validation set (Section 4.3). If the quantized model's performance is good enough w.r.t. the floating point one after repair, the model is ready for deployment. Otherwise, QNNRepair continues by selecting other parameters to repair. More detailed information is presented in Algorithm 1 (Section 4.4).

4.1 Ranking the Importance of the Neurons

QNNRepair starts with evaluating the importance of the neurons in the neural network for causing the output difference between the quantized model and the floating point one. When conducting an inference procedure on an image, the intermediate layer in the model has a series of outputs as the inputs for the next layer. The outputs go through activation functions, and we assume it is a ReLU function. For the output, if it is positive, we place it as one. If not, we place it as zero, naming it the activation output. Let f_i and q_i represent the activation output of a single neuron in full-precision and quantized models separately. If there is a testing image that makes (f_i, q_i) not equal, we consider the neuron as "activated", and we set $v_{mn} = 1$, otherwise $v_{mn} = 0$. Then we define the activation function matrix to assemble the activation status of all neurons for the floating-point model:
$\begin{pmatrix} f_{11} & \cdots & f_{1n} \\ \vdots & \ddots & \vdots \\ f_{m1} & \cdots & f_{mn} \end{pmatrix} = f_i$ and q_i for the quantized model.

We define the activation differential matrix to evaluate the activation difference between the floating point and the quantized model. Given an input image i, we calculate $\mathrm{diff}_i = f_i - q_i$ between the two models. We form a large matrix of these $diff_i$ regarding the image i. The element in this matrix should be 0 or 1, representing whether the floating and quantized neural networks' activation status is the same.

We borrow the concepts from traditional software engineering, just replacing the statements in traditional software with neurons in neural network models. We define the passing tests as the images in the repair set that the floating-point and quantized model have the same classification output, and failing tests as their classification results are different. For a set of repair images, we define $< C_n^{\mathrm{af}}, C_n^{\mathrm{nf}}, C_n^{\mathrm{as}}, C_n^{\mathrm{ns}} >$ as following:

- C_n^{af} is the number of "activated" neurons for failing tests.
- C_n^{nf} is the number of "not activated" neurons for failing tests.
- C_n^{as} is the number of "activated" neurons for passing tests.
- C_n^{ns} is the number of "not activated" neurons for passing tests.

We borrow the concepts from traditional software fault localization: Tarantula [27], Ochiai [2], DStar [47], Jaccard [3], Ample [9], Euclid [16] and Wong3 [48] and defined the indicators of neuronal suspicion in Table 1. Note that in DStar, * represents the n square of C_n^{af}.

We then rank the quantitative metrics of these neurons from largest to smallest based on certain weights, with higher metrics indicating more suspicious neurons and the ones we needed to target for repair.

Table 1: Importance (i.e., fault localization) metrics used in experiments

Tarantula:	$\frac{C_n^{\mathrm{af}}/\left(C_n^{\mathrm{af}}+C_n^{\mathrm{nf}}\right)}{\mathrm{C}_n^{\mathrm{af}}/\left(C_n^{\mathrm{af}}+C_n^{\mathrm{nf}}\right)+C_n^{\mathrm{as}}/\left(C_n^{\mathrm{as}}+C_n^{\mathrm{ns}}\right)}$	Euclid:	$\sqrt{C_n^{\mathrm{af}}+C_n^{\mathrm{ns}}}$
Ochiai:	$\frac{C_n^{\mathrm{af}}}{\sqrt{\left(C_n^{\mathrm{af}}+C_n^{\mathrm{as}}\right)\left(C_n^{\mathrm{af}}+C_n^{\mathrm{nf}}\right)}}$	DStar:	$\frac{C_n^{\mathrm{af}^*}}{C_n^{\mathrm{as}}+C_n^{\mathrm{nf}}}$
Ample:	$\left\lvert\frac{C_n^{\mathrm{af}}}{C_n^{\mathrm{af}}+C_n^{\mathrm{nf}}}-\frac{C_n^{\mathrm{as}}}{C_n^{\mathrm{as}}+C_n^{\mathrm{ns}}}\right\rvert$	Jaccard:	$\frac{C_n^{af}}{C_n^{af}+C_n^{nf}+C_n^{as}}$
Wong3:	$C_n^{\mathrm{af}}-h \;\; h=\begin{cases} C_n^{as} & \text{if } C_n^{as}\leq 2 \\ 2+0.1\left(C_n^{as}-2\right) & \text{if } 2<C_n^{as}\leq 10 \\ 2.8+0.01\left(C_n^{as}-10\right) & \text{if } C_n^{as}>10 \end{cases}$		

4.2 Constraints-solving based Repairing

After the neuron importance evaluation, for each layer, we obtain a vector of neuron importance. We rank this importance vector. The neuron with the highest importance is our target for repair and as it could have the greatest impact on the corrected error outcome.

The optimization problem for a single neuron can be described as follows:

$$\begin{aligned} \text{Minimize} &: M \\ \text{Subject to} &: M \geq 0; (\delta_1, \delta_2, \ldots, \delta_n) \in [-M, M] \\ & \forall x_i \text{ in TestSet } X\text{: } \sum_{i=1}^{m} w_i x_i < 0 \quad \text{and} \quad \sum_{i=1}^{m} (w_i + \delta_i) x_i > 0 \end{aligned} \tag{3}$$

In the formula, m represents the number of neurons connected to the previous layer of the selected neuron, and we number them from 1 to m, M is used to make $\delta_1 ... \delta_i$ are sufficiently small. We add incremental δ to the weights to indicate the weights that need to be modified, all the way to m. We assume that in the full-precision neural network, this neuron's activation function gives the result 1, and the quantized gives 0. The corrected neuron in the quantized model result needs to be greater than 0 for the output of the activation function to be 1. In this case, we make the distance of the repaired quantized neural network as close as possible to that of the original quantized neural network.

The inputs to our algorithm are a quantized neural network Q that needs to be repaired, a set of data sets X for testing, and the full-precision neuron network model F to be repaired. We use Gurobi [36] as the constraint solver to solve the constraint and then replace the original weights with the result obtained as the new weights. After the repair is complete, we re-verify the fidelity accuracy to confirm that our requirements are met. If so, we consider the repaired network to be ready to use. If not, we select the second most important neuron and perform the repair process again, until the accuracy satisfies our requirements.

4.3 Evaluating the performance similarity between the two models

The goal of quantized neural network repair is to improve the model's accuracy to be closer (and even better) than its floating-point version. Hence, the fidelity accuracy [52] is chosen to quantify the performance similarity between two neural network models. Let X denote the test set, $f_F(x)$ denotes the classification output of the floating-point model, and $f_Q(x)$ – the classification output of the quantized model. The fidelity accuracy can be defined as the following equation:

$$\text{Fidelity } = 1 - \text{Prob}\left\{x \in X \mid f_F(x) \neq f_Q(x)\right\} \tag{4}$$

In the equation, $Prob$ indicates the probability that a floating-point model and a quantized model produce different outputs for the same input image.

4.4 QNNRepair Algorithm

Our repair method is formulated in Algorithm 1. The input to our algorithm is the full-precision model F, the quantized model Q, the repair set X, the validation set V, and the number of neurons that need to be repaired N (Line 1). Firstly, we initialize arrays to store the activation states of the floating and quantized model, the values of the neuron importance, and four arrays $C^{as}[]$, $C^{af}[]$, $C^{ns}[]$ and $C^{nf}[]$ mentioned in Section 4.1 (Line 1). For these six arrays, we set all elements to 0.

Next, in lines 3-4, for each input in the test set $x \in \mathcal{X}_n$, we perform the inference process once and obtain the neurons' activation states in the corresponding model layers, and store them in the activation states of the floating and quantized model. In line 5, if $x[i]$ is a failing test, then we add the difference of activation status between the float model and quantized model to $C^{as}[i]$, and vice versa. In line 11 and 12, we calculate $C^{ns}[]$ and $C^{nf}[]$ according to the definition in Section 4.1. We calculate the importance (here we use DStar as an example) for each neuron regarding seven importance metrics and sort them in descending order then store them in set $I_n[]$ in line 14.

Then, we pick the neuron in $I_n[]$, according to the neuron's weights and the corresponding inputs from the previous layer, we create and solve the LP problem we discussed in Section 4.2, get the correction of each neuron, and update their weights. When it arrives at the maxim number of neurons to repair, the loop breaks, and we have corrected all the neurons. These are implemented at lines 17-24 in Algorithm 1.

Algorithm 1: Repair algorithm

```
Input: Floating-point model $F$, Quantized model $Q$, Repair set $X$, Validation
       set $V$, Number of neurons to be repaired $N$
Output: Repaired model $Q'$, Repaired model's accuracy $Acc$
Initialize $F_a[][], Q_a[][], I_n[]$, $C^{as}[], C^{af}[]$, $C^{ns}[], C^{nf}[]$
foreach $X$ do
    $F_a[][i] = \text{getActStatus}(F, x_i)$
    $Q_a[][i] = \text{getActStatus}(Q, x_i)$
    if $x[i]$ is a failing test then
        $C^{af}[i] = C^{af}[i] + |F_a[][i] - Q_a[][i]|$
    else
        $C^{as}[i] = C^{as}[i] + |F_a[][i] - Q_a[][i]|$
    end
end
$C^{nf}[] = C^{nf}[] - C^{af}[]$
$C^{ns}[] = C^{ns}[] - C^{as}[]$
$I_n[] = \text{DStar}(C_n^{as}[], C_n^{as}[], C_n^{as}[], C_n^{as}[])$
$I_n[] = \text{sort}(I_n[])$ // In descending order
Initialize weight of neurons $w[][]$ and the increment $\delta[][]$
foreach $neuron[i] \in I_n[]$ do
    foreach $edge[j][i] \in neuron[i]$ do
        $w[j][i] = \text{getWeight}(edge[j][i])$
    end
    $\delta[][i] = \text{solve}(X, w[][i])$ // Solve LP problem 3
    foreach $edge[j][i] \in neuron[i]$ do
        $edge[j][i] = \text{setWeight}(w[j][i] + \delta[j][i])$
        $Q' = \text{update}(Q, edge[j][i])$
    end
    if $i >= N$ then
        break
    end
end
$Acc = \text{calculateAcc}(Q', V)$
$FidelityAcc = \text{calculateFidelityAcc}(Q, Q')$
return $Q'$, $FidelityAcc$
```

Finally, we evaluate the fidelity accuracy of the corrected model. If it satisfies our requirements, then the model is repaired. Otherwise, try other combinations of parameters like important metrics or the maxim number of neurons need to repair and repeat the LP solving and correction process. The output for this algorithm is the repaired model with updated weight.

5 Experiment

5.1 Experimental Setup

We conduct experiments on a machine with Ubuntu 18.04.6 LTS OS Intel(R) Xeon(R) Gold 5217 CPU @ 3.00GHz and two Nvidia Quadro RTX 6000 GPUs. The experiments are run with TensorFlow2 + nVidia CUDA platform. We use the Gurobi [36] as the linear program solver and enable multi-thread solving(up to 16 cores). We apply QNNRepair to repair a benchmark of five quantized neural network models, including MobileNetV2 [37] on ImageNet datasets [12], and ResNet-18 [23], VGGNet [39] and two simple convolutional models trained on CIFAR-10 dataset [30]. The details of these models are given in Table 2.

We obtained the full-precision MobileNetV2 directly from Keras library, whereas we trained the VGGNet and ResNet-18 models on the CIFAR-10 dataset.

Table 2: The baseline models. Parameters include the trainable and non-trainable parameters in the models; the unit is million (M). The two accuracy values are for the original floating point model and its quantized version, respectively.

				Accuracy	
Model	Dataset	#Layers	#Params	floating point	quantized
Conv3	CIFAR-10	6	1.0M	66.48%	66.20%
Conv5	CIFAR-10	12	2.6M	72.90%	72.64%
VGGNet	CIFAR-10	45	9.0M	78.67%	78.57%
ResNet-18	CIFAR-10	69	11.2M	79.32%	79.16%
MobileNetV2	ImageNet	156	3.5M	71.80%	65.86%

We also defined and trained two smaller convolutional neural networks on CIFAR-10 for comparison: Conv3, which contains three convolutional layers, and Conv5, which contains five convolutional layers. Both models have two dense layers at the end. ~~AAAAA~~ ~~AAAA~~ ~~BBBB~~ The quantized models are generated by using TensorFlow Lite (TFlite) [1], from the floating point models. In TFLite, we chose dynamic range quantization, and the weights are quantized as 8-bit integers. The quantized convolution operation is optimized for performance, and the calculations are done in the fixed-point arithmetic domain to avoid the overhead of de-quantizing and re-quantizing tensors.

For repairs of the quantized model's performance, we use a subset of ImageNet called ImageNet-mini [26], which contains 38,668 images in 1,000 classes. The dataset is divided into the repair set and the validation set. The repair set contains 34,745 images, and the validation set contains 3,923 images. The CIFAR-10 dataset contains 60,000 images in 10 classes in total. 50,000 of them are training image, and 10,000 of them are test set. We use 1,000 images as the repair set. We use the repair set to identify suspicious neurons, generate LP constraints, and apply corrections to the identified neurons, and use the validation set to evaluate the accuracy of the models. We repeat the same experiment ten times for random neuron selection and get the average to eliminate randomness in repair methods.

5.2 Repair Results on Baselines

In this part, we apply QNNRepair to these baseline quantized models, except for MobilenetV2, in Table 2. In our experiments, MobileNetV2 is trained on ImageNet while other models are trained on CIFAR-10, and it contains more layers. The results for MobileNetV2 are reported in 5.5. For each model, we perform a layer-by-layer repair of its last dense layers. We name these dense layers as dense-3 (the third last layer), dense-2 (the second last layer), and dense-1 (the output layer).

The QNNRepair result are reported in Table 3. We ranked the neurons using important metrics and chose the best results among the seven metrics.

Table 3: QNNRepair results on CIFAR-10 models. The best repair outcome for each model, w.r.t. the dense layer in that row, is in **bold**. We further highlight the best result in *blue* if the repair result is even better than the floating point model and in red if the repair result is worse than the original quantized model. Random means that we randomly select neurons at the corresponding dense layer for the repair, whereas Fault Localization refers to the selection of neurons based on important metrics in QNNRepair. In All cases, all neurons in that layer are used for repair. 'n/a' happens when the number of neurons in the repair is less than 100, and '-' is for repairing the last dense layer of 10 neurons, and the result is the same as the All case.

	Random				Fault Localization				-
#Neurons repaired	1	5	10	100	1	5	10	100	All
Conv3_dense-2	63.43%	64.74%	38.90%	n/a	66.26%	**66.36**%	62.35%	n/a	57.00%
Conv3_dense-1	65.23%	66.31%	-	n/a	66.10%	66.39%	-	n/a	**66.46**%
Conv5_dense-2	72.49%	72.55%	72.52%	72.52%	**72.56**%	72.56%	72.56%	72.56%	72.54%
Conv5_dense-1	72.51%	72.52%	-	n/a	**72.58**%	72.56%	-	n/a	72.56%
VGGNet_dense-3	78.13%	78.44%	78.20%	78.38%	**78.83**%	78.82%	78.78%	78.66%	78.60%
VGGNet_dense-2	78.36%	78.59%	78.44%	78.22%	78.55%	**78.83**%	78.83%	78.83%	78.83%
VGGNet_dense-1	78.94%	67.75%	-	n/a	**79.29**%	69.04%	-	n/a	74.49%
ResNet_dense_1	78.90%	78.92%	-	n/a	79.08%	**79.20**%	-	n/a	78.17%

We also run randomly picked repairing as a comparison. We have chosen Top-1, Top-5, Top-10, Top-100, and all neurons as the repairing targets. For most models, the repair improves the accuracy of the quantized network, and in some cases, even higher than the accuracy of the floating-point model.

The dense-2 layer only contains 64 neurons in the Conv3 model. Hence we selected 30 neurons as the repair targets. In the dense-1 layer of Conv3, the effect of repairing individual neurons is not ideal, but as the number of repaired neurons gradually increases, the more correct information the Conv3 quantization model obtains from the floating-point model, so the accuracy gradually improves until it reaches 66.46% (which does not exceed the accuracy of the floating-point Conv3 neural network, but it gets very close to it: 66.48%, see Table 2). This is because all the repair information in the last layer comes from the original floating-point neural network. Note that because of the simple structure of the Conv3 neural network, the floating-point version of Conv3 itself is inaccurate, and the quantized and repaired neural network does not exceed the accuracy. In the dense-2 layer of Conv5, applying importance metrics to repair this layer is slightly better than random selection, only 0.01% regarding randomly selecting 5 neurons compared with using fault localization to select Top-5 neurons. Compared to the quantized model before repair, whose accuracy is 72.64%, the repairing only gets an accuracy of 72.56%, which does not improve the model's accuracy. In the dense-1 layer of Conv5, the best result is using fault localization to pick the Top-1 neuron and repair, at 72.58% accuracy, and this is not better than the quantized model before repair.

For VGGNet and ResNet-18 neural networks, the dense-1 layer is a good comparison. Both VGGNet and ResNet-18 have relatively complex network structures, and the accuracy of the original floating-point model is close to 80%. In the dense-1 layer of ResNet-18, only some of the neurons were repaired with accuracy close to their original quantized version, but all of them did not exceed the exact value of the floating-point neural network after the repair. However, unlike ResNet-18, correcting a single neuron randomly in the dense-1 layer of VGGNet make it more accurate than the quantized version of VGGNet. Using the importance metric and correcting a single neuron make the accuracy even higher than the floating-point version of VGGNet. However, repairing dense-1 of VGGNet was unsatisfactory, especially when 5 neurons are selected for repair; it suffered a significant loss of accuracy, even below 70%, which was regained if all ten neurons in the last layer were repaired. In the dense-2 layer of VGGNet, the overall accuracy is higher than 78%. When the importance metric is applied, the accuracy reaches 78.83%, noting that this accuracy is also achieved if all neurons in this layer are repaired. For the dense-3 layer of VGGNet, repairing 5 or 10 neurons using importance metrics will achieve the highest accuracy at 78.83%, the same as repairing the dense-2 layer.

Table 4: QNNRepair results on ImageNet model.

	Random		Fault Localization		-
#Neurons repaired	10	100	10	100	All
MobileNetV2_dense-1	70.75%	70.46%	**70.77%**	70.00%	68.98%

ImageNet We also conducted repair on the last layer for MobileNetV2 trained on the ImageNet dataset of high-resolution images. Using Euclid as the importance metric and picking 10 neurons as the correct targets achieve the best results, at 70.77%, improving the accuracy the quantized model.

5.3 Comparison with Data-free Quantization

We tested SQuant [21], a fast and accurate data-free quantization framework for convolutional neural networks, employing the constrained absolute sum of error (CASE) of weights as the rounding metric. We tested SQuant two quantized models, the same as our approach: MobileNetV2 trained on ImageNet and ResNet-18 on CIFAR-10. We made some modifications to the original code to support MobileNetV2, which is not reported in their experiments.

In contrast, to complete data-free quantization, our constraint solver-based quantization does not require a complete dataset and but only some input images for repair. Despite taking much more time than SQuant because it uses Gurobi and a constrained solution approach, MobileNetV2 – a complex model trained on ImageNet – QNNRepair achieves much higher accuracy.

Table 5: QNNRepair vs SQuant

	MobileNetV2		ResNet-18	
	Accuracy	Time	Accuracy	Time
SQuant [21]	46.09%	1635.37ms	70.70%	708.16ms
QNNRepair	**70.77%**	∼15h	**79.20%**	∼9h

5.4 Repair Efficiency

The constraints-solving part contributes to the major computation cost in QNNRepair. For other operations, such as importance evaluation, modification of weights, model formatting, etc., it takes only a few minutes to complete. Thereby, Table 6 measures the runtime cost when using the Gurobi to solve the values of the new weights for a neuron for our experiments on the VGGNet model. It is shown in Table 6 that 75% of the solutions were completed within 5 minutes, and less than 9% of the neurons could not be solved, resulting in a total solution time of 9 hours for a layer of 512 neurons.

Table 6: The Gurobi solving time for constraints of each neuron in the dense-2 layer of the VGGNet model. There are 512 neurons in total.

Duration	<=5mins	5-10mins	10-30mins	30mins-1h	No solution
Percentage	75%	8.98%	5.27%	1.76%	8.98%

5.5 Comparison Between Fault Localization Metrics in QNNRepair

We let the model and the layer stay the same. We use MobileNetV2 and the last layer as our target. We compare seven representative important metrics mentioned in Section 5.5. In these experiments, we used 1000, 500, 100, and 10 jpeg images as the repair sets to assess the performance of different importance assessment methods.

Firstly we rank the neurons in the last layer using seven different representative important metrics, which are Tarantula [27], Ochiai [2], DStar [47], Jaccard [3], Ample [9], Euclid [16] and Wong3 [48]. As shown in Figure 2, for the last fully connected layer of MobileNetV2, the important neurons are mainly concentrated at the two ends, those neurons with the first and last numbers. The evaluation metrics results are relatively similar for different neurons.

We selected the 100 neurons (for Conv3 it is 30 neurons) with the highest importance and could be solved by linear programming solvers according to different importance measures. The deltas are obtained according to Equation 3, and we apply the deltas to the quantized model. After that, we use the validation

Table 7: The results regarding importance metrics, including 7 fault localization metrics and 1 random baseline. The number of images indicates how many inputs are in the repair set.

Model+Repair Layer	#Images	Tarantula	Ochiai	DStar	Jaccard	Ample	Euclid	Wong3	Random
MobileNetV2_dense-1	1000	70.61%	69.76%	69.73%	69.73%	69.72%	**70.70**%	69.73%	69.56%
	500	68.99%	69.01%	69.05%	69.05%	68.99%	**69.46**%	69.06%	69.00%
	100	69.50%	69.42%	69.46%	69.46%	69.53%	**69.98**%	69.46%	70.12%
	10	**70.62**%	70.15%	70.12%	70.12%	70.17%	70.73%	70.12%	70.18%
VGGNet_dense-3	1000	78.64%	78.64%	78.64%	78.64%	78.65%	**78.66**%	**78.66**%	78.22%
VGGNet_dense-2	1000	78.83%	78.83%	78.83%	78.83%	78.83%	78.83%	78.83%	**78.38**%
Conv3_dense	1000	**59.50**%	**59.50**%	**59.50**%	**59.50**%	59.27%	59.27%	59.27%	32.42%

sets from ImageNet, which contains 50,000 jpeg images, to test the MobileNetV2 model. We also use the validation sets from CIFAR-10, which contains 10000 png image files, to test VGGNet and Conv5 after the repair. As a comparison, we also randomly picked 100 neurons to apply to repair and tested their accuracy. We give the results of the top 100 important neurons after selection and repair, as shown in Table 7.

We pick Tarantula, the best result of using ten images to repair the MobileNetV2 last layer, and plot the scatter plots based on the importance distribution of the different neurons. We rank the importance of those neurons and draw line plots as illustrated in Figure 2.

The figures give scatter plots of neuron importance and ranked line plots for the last dense layer of MobileNetV2. The horizontal coordinates of these plots are the serial numbers of the neurons. For the last layer in the MobileNetV2 model, few neurons have the highest importance. More than 300 neurons had an importance measurement of 0, and another large proportion had an importance of 0.5 or less. Based on the ranking of the importance of neurons, all the evaluation metrics except Tarantula and Euclid considered 108, 984, 612, 972 to be the four most important neurons in this layer, and among the 5th-10th most important neurons, 550, 974, 816, 795 and 702, just in a different order. This is reflected in the importance distribution graphs as spikes at the ends and as spikes at the ends of the graphs. Hence Ochiai, Dstar, Jaccard, Ample, and Wong3 have similar performance regarding the accuracy evaluation, and Euclid and Tarantula achieve better accuracy on ImageNet validation sets. Table 7 shows that the Euclid importance assessment method is highly effective, achieving relatively good results from restoration with 500 images to restoration with ten images and achieving only weaker accuracy than the Tarantula method in a restoration scenario with 1000 images. Also, a random selection of neurons can achieve good restoration results, especially when we select 100 images as restoration images. Only the random selection method has a validation accuracy higher than 70%.

For VGGNet model, for the same reason as MobileNetV2, the Tarantula, Ochiai, DStar, Jaccard, Euclid, and Wong3 give the same results when selecting 100 top important neurons to repair. As a comparison, the accuracy of random selection in dense-2 layer and dense-3 has a slight drop, at 78.38% and 78.22%.

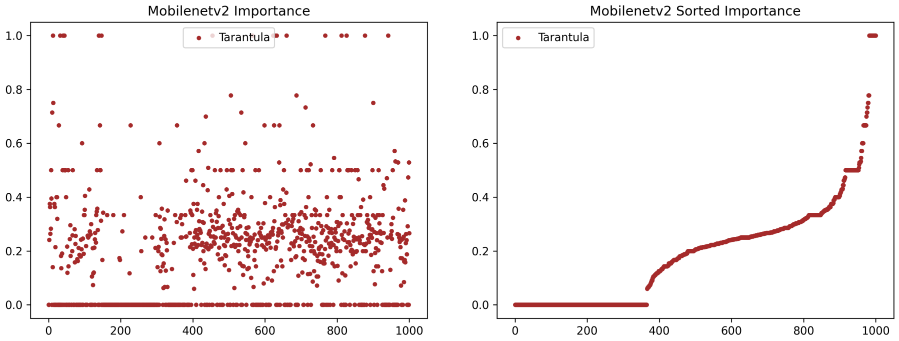

Fig. 2: Importance distribution regarding certain importance metrics on MobileNetV2.

For Conv3 model, the seven importance metrics give the same results, and randomly select 30 neurons suffered a great accuracy loss, at 32.42%. But compared to the results in Table 3, repairing 30 top neurons also suffered accuracy drops. For the dense layer of conv3, the best repair is still to select one neuron for repair based on Tarantula sorting at 66.10%, and if random selection is taken into account, then selecting five neurons for repair would give the best result at 64.74%.

We also conducted a side-by-side comparison of the number of images required for the repair on MobileNetV2. It shows that the best results are obtained using 1000 images for repair and 10 images for the repair, but given the amount of time required to generate constraints for the repair using 1000 images and to solve the constraints using Gurobi, we recommend using a smaller set of repair images for the model.

Euclid demonstrates that it has the highest accuracy most of the time, and repairing with importance evaluation is more accurate than repairing randomly selected neurons.

5.6 Limitations

For larger models, it takes a longer time to solve the linear programming problem. Also, selecting more repairing images for correction will have a greater likelihood of Gurobi being unable to solve the linear programming problem, reflected in the limitation of improving accuracy.

6 Conclusion

In this paper, we presented QNNRepair, a novel method for repairing quantized neural networks. Our method is inspired by traditional software statistical fault localization. We evaluated the importance of the neural network models and used Gurobi to get the correction for these neurons. According to the experiment results, after correcting the model, accuracy increased compared with the quantized model. We also compared our method with state-of-the-art techniques; the experiment results show that our method can achieve much higher accuracy when repair models are trained on large datasets.

7 Future Works

We will move forward to larger datasets; currently, we support MobileNetV2 trained on ImageNet. In the future, we will test our tool and make it scalable for larger models and not limited to classification tasks like GPT and stable diffusion. For these large networks, due to the complexity of the model itself, repairing them will require a lot of computational resources, and we will find a balance between improving accuracy and computing time.

For some of the repairing problems, Gurobi was not able to solve them in the given time limit, so in the future, we intend to optimize the encoding of the neural network repair problem to increase the speed of the repair solution and to solve some of the repair problems that were not previously solved. We will also try more problem solvers in the future, such as SMT solvers, to solve these problems that Gurobi cannot solve.

Acknowledgements

This work is funded by the EPSRC grant EP/T026995/1 *"EnnCore: End-to-End Conceptual Guarding of Neural Architectures"*. The Dame Kathleen Ollerenshaw Fellowship of The University of Manchester supports M. A. Mustafa. This work was also partially supported by Cal-Comp Electronic by the R&D project of the Cal-Comp Institute of Technology and Innovation.

References

1. TensorFlow Lite, `https://www.tensorflow.org/lite`
2. Abreu, R., Zoeteweij, P., Van Gemund, A.J.: On the accuracy of spectrum-based fault localization. In: Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007). IEEE (2007)
3. Agarwal, P., Agrawal, A.P.: Fault-localization techniques for software systems: A literature review. ACM SIGSOFT Software Engineering Notes **39**(5), 1–8 (2014)
4. Amir, G., Wu, H., Barrett, C., Katz, G.: An smt-based approach for verifying binarized neural networks. In: Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS. Springer (2021)
5. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (vnn-comp 2021): Summary and results. arXiv preprint arXiv:2109.00498 (2021)
6. Borkar, T.S., Karam, L.J.: Deepcorrect: Correcting dnn models against image distortions. IEEE Transactions on Image Processing **28**(12), 6022–6034 (2019)

7. Bressert, E.: Scipy and numpy: an overview for developers (2012)
8. Choi, J., Wang, Z., Venkataramani, S., Chuang, P.I.J., Srinivasan, V., Gopalakrishnan, K.: Pact: Parameterized clipping activation for quantized neural networks. arXiv preprint arXiv:1805.06085 (2018)
9. Dallmeier, V., Lindig, C., Zeller, A.: Lightweight bug localization with ample. In: Proceedings of the sixth international symposium on Automated analysis-driven debugging. pp. 99–104 (2005)
10. David, R., Duke, et al.: Tensorflow lite micro: Embedded machine learning for tinyml systems. Proceedings of Machine Learning and Systems (2021)
11. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: CAV. Springer (2008)
12. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. IEEE (2009)
13. Eén, N., Sörensson, N.: An extensible sat-solver. In: Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003. Springer
14. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Automated Technology for Verification and Analysis: 15th International Symposium, 2017. pp. 269–286. Springer (2017)
15. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Computer Aided Verification: 32nd International Conference, CAV 2020. pp. 43–65. Springer (2020)
16. Galijasevic, Z., Abur, A.: Fault location using voltage measurements. IEEE Transactions on Power Delivery **17**(2), 441–445 (2002)
17. Gehr, T., Mirman, M., Drachsler-Cohen, D., Others: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE symposium on security and privacy (SP). pp. 3–18. IEEE (2018)
18. Giacobbe, M., Henzinger, T.A., Lechner, M.: How many bits does it take to quantize your neural network? In: 26th International Conference, TACAS. Springer (2020)
19. Goldberger, Ben, e.a.: Minimal modifications of deep neural networks using verification. In: LPAR. p. 23rd (2020)
20. Gong, Ruihao, e.a.: Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4852–4861 (2019)
21. Guo, Cong, e.a.: Squant: On-the-fly data-free quantization via diagonal hessian approximation. arXiv preprint arXiv:2202.07471 (2022)
22. Guo, Y.: A survey on methods and theories of quantized neural networks. arXiv preprint arXiv:1808.04752 (2018)
23. He, Kaiming, e.a.: Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition pp. 770–778 (2016)
24. Henzinger, T.A., Lechner, M., et al.: Scalable verification of quantized neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence (2021)
25. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Computer Aided Verification: 29th International Conference, CAV 2017. pp. 3–29. Springer (2017)
26. ifigotin: Imagenetmini-1000. `https://www.kaggle.com/datasets/ifigotin/imagenetmini-1000` (2021), accessed: April 4, 2023
27. Jones, J.A., Harrold, M.J.: Empirical evaluation of the tarantula automatic fault-localization technique. In: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. pp. 273–282 (2005)

28. Katz, G., Barrett, C., Others: Reluplex: An efficient smt solver for verifying deep neural networks. In: Computer Aided Verification: 29th International Conference, CAV 2017. Springer (2017)
29. Katz, G., Huang, D.A., Ibeling, D., et al.: The marabou framework for verification and analysis of deep neural networks. In: CAV. Springer (2019)
30. Krizhevsky, Alex, Hinton, Geoffrey: CIFAR-10 (canadian institute for advanced research). Tech. rep., University of Toronto (2009), `https://www.cs.toronto.edu/~kriz/cifar.html`
31. Li, Yuhang, e.a.: Brecq: Pushing the limit of post-training quantization by block reconstruction. arXiv preprint arXiv:2102.05426 (2021)
32. Makhorin, A.: Glpk (gnu linear programming kit). http://www. gnu. org/s/glpk/glpk. html (2008)
33. Meindl, B., Templ, M.: Analysis of commercial and free and open source solvers for linear optimization problems. Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS **20** (2012)
34. Naish, L., Lee, H.J., Ramamohanarao, K.: A model for spectra-based software diagnosis. ACM Transactions on software engineering and methodology (TOSEM) **20**(3), 1–32 (2011)
35. Nickel, S., Steinhardt, C., Schlenker, H., Burkart, W.: Ibm ilog cplex optimization studio—a primer. In: Decision Optimization with IBM ILOG CPLEX Optimization Studio: A Hands-On Introduction to Modeling with the Optimization Programming Language (OPL), pp. 9–21. Springer (2022)
36. Optimization, G.: Inc.. gurobi optimizer reference manual, version 5.0 (2012)
37. Sandler, M., e.a.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition (2018)
38. Sena, L.H., Song, X., da S. Alves, E.H., Bessa, I., Manino, E., Cordeiro, L.C.: Verifying quantized neural networks using smt-based model checking. CoRR **abs/2106.05997** (2021), `https://arxiv.org/abs/2106.05997`
39. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
40. Song, C., Fallon, E., Li, H.: Improving adversarial robustness in weight-quantized neural networks. arXiv preprint arXiv:2012.14965 (2020)
41. Song, X., Manino, E., Sena, L.H., da S. Alves, E.H., de Lima Filho, E.B., Bessa, I., Luján, M., Cordeiro, L.C.: Qnnverifier: A tool for verifying neural networks using smt-based model checking. CoRR **abs/2111.13110** (2021), `https://arxiv.org/abs/2111.13110`
42. Song, X., Sun, Y., Mustafa, M.A., Cordeiro, L.: Airepair: A repair platform for neural networks. In: ICSE-Companion. IEEE/ACM (2022)
43. Sotoudeh, M., Thakur, A.V.: Provable repair of deep neural networks. In: PLDI (2021)
44. Usman, M., Gopinath, D., Sun, Y., Noller, Y., Păsăreanu, C.S.: Nnrepair: Constraint-based repair of neural network classifiers. In: CAV. Springer (2021)
45. Vanholder, H.: Efficient inference with tensorrt. In: GPU Technology Conference. vol. 1, p. 2 (2016)
46. Wang, S., Zhang, H., Xu, K., Others: Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. Advances in Neural Information Processing Systems **34** (2021)
47. Wong, W.E., Debroy, V., Gao, R., Li, Y.: The dstar method for effective software fault localization. IEEE Transactions on Reliability **63**(1), 290–308 (2013)

48. Wong, W.E., Qi, Y., Zhao, L., Cai, K.Y.: Effective fault localization using code coverage. In: COMPSAC. vol. 1, pp. 449–456. IEEE (2007)
49. Yu, Bing, e.a.: Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment. IEEE Transactions on Reliability (2021)
50. Zhang, Yedi, e.a.: Qvip: an ilp-based formal verification approach for quantized neural networks. In: ASE. IEEE/ACM (2022)
51. Zhang, J., Li, J.: Testing and verification of neural-network-based safety-critical control software: A systematic literature review. Information and Software Technology **123**, 106296 (2020)
52. Zhou, Z.H.: Rule extraction: Using neural networks or for neural networks? Journal of Computer Science and Technology **19**(2), 249–253 (2004)