Association Rule Mining Based Algorithm for Recovery of Silent Data Corruption in Convolutional Neural Network Data Storage

Mohammadreza Ramzanpour
Department of Computer Science
North Dakota State University
Fargo, USA
m.ramzanpour@ndsu.edu

Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, USA
simone.ludwig@ndsu.edu

***Abstract*—Embedded systems are finding their way into almost every aspects of our daily life from mp3 players and console games to the mobile phones. Different Artificial Intelligence (AI) based applications are commonly utilized in embedded systems from which computer vision based approaches are included. The demand for higher accuracy in computer vision applications is associated with the increased complexity of convolutional neural networks and the storage requirement for saving pre-trained networks. Different factors can lead to the data corruption in the storage units of the embedded systems, which can result in drastic failures due to the propagation of the errors. Hence, the development of software-based algorithms for the detection and recovery of data corruption is crucial for improvement and failure-prevention of embedded systems. This paper proposes a new algorithm for the recovery of the data in the case of single event upset (SEU) error. The association rule mining based algorithm will be used to find the probability of the corruption in each of the bits. The recovery algorithm was tested on four different pre-trained ResNet (ResNet32 and ResNet110 at two different accuracy levels each) and the best recovery rate of 66% was found in the most complex scenario, i.e., random bit corruption. However, for the special cases of SEU errors, e.g. error in the frequently repeated bits, the recovery rate was found to be perfect with a value of 100%.**

***Index Terms*—Single event upset, Silent data corruption, Association rule mining, Recovery algorithm**

I. Introduction

Artificial intelligence (AI) applications are increasingly growing in different fields including the ones used in our every-day lives. Historically speaking, the attempts on the AI side were focused on mimicking human behavior up to the level where one cannot recognize the computer performance from a human. However, most of the attempts in this area shifted to the implementation of rational thinking instead. Nowadays, AI plays role in robotics, speech recognition, natural language processing (NLP), data driven modeling, recommendation systems, and computer vision. Each of the aforementioned categorical fields can be even sub-divided further to list more specific applications which have found their way into different technological and industrial applications.

Prediction systems which are commonly used in AI, are mostly constructed by utilizing machine learning (ML) techniques. From one point of view, ML techniques can be categorized into unsupervised and supervised learning. The unsupervised ML algorithms are aimed at assigning instances within a set of unlabeled data into different groups (clusters). Usually due to the lack of appropriate pre-obtained information, it is hard to evaluate and assess the quality and efficiency of the unsupervised algorithms. On the other hand, supervised ML techniques are generated by training a probabilistic hypothesis on a set of annotated/labeled data to be further used for prediction of unseen data. Other recently introduced and used ML approaches such as reinforcement learning and semi-supervised learning can be generalized as one or a hybrid mix of the aforementioned approaches.

One of the very common and popular ML techniques that has gained the attraction of the researchers in different fields is artificial neural networks (ANN). Inspired by biological neurological networks, they have been used for both unsupervised and supervised learning [1], [2]. ANN has a layered structure in which the output of one layer will be the input for the next layer. The size of each layer is determined by the number of its activation units (which resembles the neuron synapses in equivalent biological network). Using multiple layers with multiple activation units yields a highly nonlinear function, which could be a very powerful tool for dealing with complex problems. In the AI community and the literature, the approaches involved with ANNs are also called deep learning (DL) methods, which refers to their layered structure.

Convolutional neural networks (CNN) is a DL approach, which is commonly used in computer vision applications such as object classification [3], object detection [4], and semantic segmentation [5], [6]. CNN is based on a mathematical operation named as convolution, which is accomplished by the use of kernels (filters). While these kernels can be hand engineered to find certain useful features in the images (like horizontal or vertical edges), kernel parameters can be trained in a supervised manner to get tailored for specific applications. One advantage of using kernels in CNN is that its number of parameters is not related to the input size while this is not the case in fully connected networks. However, CNNs are

TABLE I
SOME OF THE FAMOUS CNN ARCHITECTURES WITH THE NUMBER OF PARAMETERS INVOLVED.

Architecture name	Number of parameters
AlexNet [7]	60M
VGG [8]	138M
GooLeNet [9]	4M
DenseNet [10]	15M-25M

commonly accompanied by several fully connected layers for performance improvement. In many of the CNN architectures, these layers account for an reasonable fraction of the total number of parameters. As ML problems get more challenging, the number of layers or number of kernels to be used in the CNN increases, which consequently results in a large number of parameters. The number of parameters involved in some of the famous CNN architectures are demonstrated in Table I. As it can be seen from Table I, the number of parameters ranges from hundreds of thousands to hundred millions. Therefore, it is no surprise that the optimization involved in training of such a networks requires large computational power. The appearance of graphical processing units (GPU) and appropriate frameworks such as Compute Unified Device Architecture (CUDA) provides a powerful computational capability for this purpose. While the training phase is slow and computationally expensive, the inference (prediction) phase is relatively much faster since it is basically a function evaluation.

With the ever-increasing challenges in the computer vision field, the new methodologies appeared in the architecture of the CNNs, which in most of the cases resulted in the generation of deeper networks, i.e., utilization of more layers. However, the dramatic increase in the depth of the network causes an issue in the training phase known as the vanishing gradient problem. Even the use of augmented layers like inception networks [9], which combines multiple kernels and pooling layers into one, suffers from the vanishing gradient. The ResNet architecture originally introduced by He et al. [11] gets around this problem by using skip connections passing through two or three layers. The batch normalization is also commonly used in ResNet to resolve the covariate shift problem [12]. As a result, the ResNet architecture can go deep to the order of hundreds of layers while keeping the number of trainable parameters relatively low [13].

Embedded system is a term referring to a category of computing based devices that can be a combination of hardware an software in the context of electrical or even mechanical devices. The concept of embedded systems can be seen in different devices, such as mobile phones, digital watches, electronic vehicles, medical imaging devices, etc. Computer vision applications are increasingly implemented in the embedded devices [14] such as autonomous cars, smart home security, and facial recognition gadgets commonly available these days in different mobile phones.

Certain constraints such as size limitation, power supply, and harsh operation environments may affect the performance of the embedded systems [15], which can lead to the introduction of soft errors in RAM, CPU and/or other computing or storage units of those systems. Different theories have been introduced as a reason behind the soft errors from which the alpha particles [16], high energy neutron from cosmic radiations [17], and low energy cosmic neutron reaction with Boron in integrated circuits (IC) [18] can be mentioned. The soft errors can be divided into two types of Single Event Upset (SEU) and burst errors. The SEU errors refer to the cases where the value of only one bit is flipped (a bit with a value of 1 gets corrupted to a value of 0 or vice versa), while in the burst error case, multiple bits (usually consecutive ones) get corrupted and their values changes. Using parity and/or checksum bits can be useful in detecting if SEU errors has happened, but it will not yield the corrupted bit and therefore, it is not possible to fix the corruption unless mirrored/backed-up data is used. While SEU only affects one bit which may result in a minor resultant error, but the cascading effect of that error, may result in the system to stop working. However, in the initial phase of this error, this error may not be noticed and this is why in the literature, it is also referred to as silent data corruption (SDC).

In this study, we develop and test an algorithm for detecting the corrupted bit in the case of SEU. Association rule mining (ARM) will be used to find the frequent patterns in the most significant bits (in 32 bit binary float representation) of the stored data. The probabilistic measures are calculated based on the confidence level criteria to calculate the probability of corruption for each bit, and consequently, the bit with the highest probability of corruption will be determined as the corrupted bit.

II. RELATED WORK

Many attempts have been made to make and improve the hardware for better resiliency against data corruption. However, this problem still cannot be avoided completely [19] and hence, the need for software-based detection and recovery of the corrupted data is a necessity.

The research toward mitigating the soft errors have been conducted in different fields. Fiala et al. [20] developed their RedMPI framework for detecting and correcting the data transferred in high-end clusters through message passing interface (MPI) systems. RedMPI achieves this by creating a replica of the primary messages via online verification of the messages and hashing techniques whereby the message sent from multiple senders is compared in order to detect the errors and prevent the spread of the data corruption.

Zick et al. [21] developed a method for detecting silent data corruption in embedded system of the NASA's cube. The first step in their method is the range checking of the processed data. If the parameters are noticed to be out of the anticipated range, that would be identified as the potential error. Since the data processing in the embedded system of their study was of the iterative type, by utilization of the data copies in the register and cache, the authors performed an extra iteration with the known input and output values to find any deviation. They refer to their approach as a built-in self-test (BIST). The

detection rate was reported to be in the range of 89 to 97% with an algorithm overhead complexity of 1%.

Charyyev et al. [22] studied different integrity verification algorithms and introduced a Robust Integrity Verification Algorithm (RIVA) for silent data corruption detection in file transfers. End-to-end integrity verification is done by calculating the checksum of the source and the copied file through hashing techniques. The calculated checksum will then be exchanged between the source and the destination server to check for any inequality. Since their proposed architecture, RIVA, is designed for file transfer in distributed systems, it consists of three concurrent threads responsible for transfer, cache evicting, and checksum calculation. In the case where a mismatch is observed between the checksum of the source and the destination file, it is assumed that the file sent by the source is correct and no approach is suggested on recovering the possible error in the source file itself.

Ni et al. [23] proposed a software based approach named FlipBack, for guarding the applications against the silent data corruption. In their approach, detection of the soft error in the field data (input data used in scientific computations) is based on the continuity of the data to detect the possible anomalies. In their analysis, spatial and temporal similarity in the data is the foundation of finding any anomaly that may have been caused by soft error. The accuracy of their recovery algorithm was found to be in the range of 80 to 100%, with an overhead complexity of 6 to 20%.

Berrocal et al. [24] used a data flow approach in the high performance computing (HPC) applications to check for deviations, which could be a potential soft error. Their proposed methodology for iterative based HPC applications consists of two steps. By time series analysis of each data point, the authors predict the expected value of that point in the consequent steps. In the next step, the normal interval range of the prediction will be calculated. The observed values lying outside this range would be assumed to be caused by soft errors. It is worth to mention that one of the predictor models the authors have used is the acceleration based predictor. In this model, the point data in the last two iterations will be used to calculate the rate of change in the point value, i.e. velocity, and the rate of change in the velocity, i.e. acceleration, to form an equation for estimating the variable point value in the following iterations. To check for the performance of their method, the testing was done on two different HPC applications. While the error prediction rate of 66% to 90% was noticed for errors introduced in these applications, no methodology was proposed for recovering the errors, which is the aim of the proposed work in this paper and is outlined in the following sections.

III. Methodology

The Apriori algorithm introduced by Agrawal and Srikant [25] is one of the most commonly used mining algorithms. The Apriori algorithm is based on the fact that if an itemset consisting of k items is frequent, then each subset of that itemset must be frequent as well. In an iterative incremental fashion, this algorithm first finds the single frequent items. In the subsequent iterations, it merges the previously found frequent items to find the potential frequent candidates and verifies if they are frequent or not. The algorithm stops at the k^{th} iteration if no itemset with the size of k can be found to be frequent. There are some disadvantages associated with the Apriori algorithm. The transaction database needs to be completely checked for finding if an itemset is frequent or not. Moreover, many spurious candidate frequent itemsets may be generated, which could be be problematic in terms of memory usage.

Bit #0	Bit #1	Bit #2	Bit #3	Bit #4	Bit #5	Bit #6	Bit #7	Bit #8
1	0	0	1	1	0	1	1	0

Fig. 1. A sample corrupted weight with 9 most significant bits ($n = 9$) with the corruption at the bit#3

The frequent pattern growth (FP-growth) algorithm introduced by Han et al. [26] uses a tree based structure for storing the database items. This algorithm uses a recursive elimination approach by removing the non-frequent individual items and all the least frequent items. While the generation of the FP-tree can be computationally expensive, the notable advantage of this algorithm is that it only requires to read the database twice. The utilization of a FP-tree for storing the frequent patterns and prefix analysis leads to a memory-efficient algorithm.

In this paper, we have used the Apriori algorithm for the ResNet32 and the FP-growth algorithm for the ResNet110 rule mining, since the ResNet110 has a larger number of parameters compared to ResNet32. It should be noted that the final result is not dependent on the rule mining algorithm and the resource constraints led us to such a decision.

The pseudo-code for our proposed recovery algorithm is shown in Algorithm 1 and here, we delineate on different aspects of this algorithm. First of all, the weights of the trained CNN (ResNet32 and ResNet110) will be flattened into a list of weights in floating point format. Thereafter, the weights will be converted into their corresponding 32-bit binary representation based on the IEEE754 standard [27]. At this stage, we have a $N \times 32$ matrix where N refers to the total number of weights of the model. We select n first bits of the binary representation as the most significant bits, and therefore, the weights matrix will shrink into a $N \times n$ matrix. This binary matrix can be viewed as a one-hot encoded list of N transactions with n items, where 1 in the cell ij refers to the presence of item j in transaction i and vice-versa for the value of 0.

The corruption prediction is based on the assumption that the corruption happens at only one specific bit of a single weight. A sample corrupted weight is shown in Figure 1 with $n = 9$, i.e., 9 significant bits where the corruption has taken place in the bit#3.

For each bit of a parameter (weight), the frequent patterns and rule mining must be done twice for each bit leading to $2n$ times of the rule mining. One time with the assumption that the bit in consideration has the value of 1 and another time for the value of 0. The bit being examined will be

Algorithm 1 The pseudo-code of the recovery algorithm.

Require: Parameters converted into 32-bit binary format
  for each bit B in msb **do**
    $i \leftarrow 0$
    $Consequent\ bit\ (cb) \leftarrow B$
    $Antecedent\ bits\ (ab) \leftarrow$ other bits
    for each $b1$ in ab **do**
      if $b_1 = 0$ **then**
        Flip all the bits with the same bit number as b_1 in the whole data.
      end if
    end for
    Run frequent pattern finding and association rule mining to find rules with their corresponding confidence values.
    $S_1 \leftarrow$ Summing over confidence values of rules where they have only B as the consequent bit.
    Flip all bits with the same bit number as B in original data.
    Run frequent pattern finding and association rule mining to find rules with their corresponding confidence values.
    $S_0 \leftarrow$ Summing over confidence values of rules where they have only B as the consequent bit.
    flip back B
    Creat an array $Corruption\ Probability\ (CP)$
    if $B = 0$ **then**
      CP(i) = S1 / S0
    else
      CP(i) = S0 / S1
    end if
    $i \leftarrow i + 1$
  end for
  $Identified\ corrupted\ bit\ (ICB) = argmax(CP)$
  return ICB

considered as the consequent bit and all the other bits will be used as the antecedent bit for the association rule mining. Therefore, for each bit, the confidence of appearance of that bit as the consequent bit will be assessed for both values of 0 and 1. If the bit in this study has the value of 1, the ratio of the confidence of the same bit with the value of 0 to its confidence when having the value of 0, will be considered as its corruption likelihood and vice versa for the time the bit has the value of 0. The bit with the highest corruption likelihood will be determined as the corrupted bit. In this paper, we have used the minimum support threshold of 0.3 for finding the frequent patterns and the confidence metric with the value of 0.3 was used for mining the rules as well. These values were fine-tuned based on our conducted experiments for achieving the best results. For finding the frequent patterns, we have used the apriori [25] or fpgrowth [26] algorithms. Apriori was used for the smaller CNNs and the fpgrowth was used for the larger CNNs with more weights, since apriori requires larger amount of memory. However, the speed of the apriori algorithm was found to be higher compared to the fpgrowth algorithm. It is worth mentioning that the number

Antecedent bits	Consequent bit	Confidence
{b_1=0}	{b_0=1}	0.5297
{b_3=0}	{b_0=1}	0.5281
{b_4=1}	{b_0=1}	0.5299
{b_1=0, b_3=0}	{b_0=1}	0.5281
{b_1=0, b_4=1}	{b_0=1}	0.5299
{b_3=0, b_4=1}	{b_0=1}	0.5281
{b_1=0, b_3=0, b_4=1}	{b_0=1}	0.5281
{b_1=0}	{b_0=0}	0.4703
{b_4=1}	{b_0=0}	0.4701
{b_1=0, b_4=1}	{b_0=0}	0.4701

Fig. 2. A sample rule mining for bit#0 as the consequent bit

Corruption likelihood								
Bit#0	Bit#1	Bit#2	Bit#3	Bit#4	Bit#5	Bit#6	Bit#7	Bit#8
0.38	0.00	2.42	2.78	0.00	0.84	0.84	0.20	0.59

Fig. 3. Calculated corruption likelihood for bits of weight shown in Fig. 1.

of most significant bits can be adjusted and even all the 32 bits could be used in the recovery algorithm. However, as we move toward the rightmost bits, the corruption would have more negligible effects on the value of the weight. Moreover, the time complexity and computational cost of the frequent pattern mining will increase dramatically as the number of most significant bits increases.

To clarify further, consider the example shown in Figure 1 where the bit number three is corrupted but this is not known to us. For the start, bit#0 will be used for rule mining. In the first step, association of bit#0 with other values must be checked by finding the frequent patterns. The association of bit#0 as the consequent bit (with the values of 1 and 0), with other bits in the weights of the CNN is shown in Figure 2.

Since, the original value of bit#0 (Figure 1) is 1, its corruption likelihood will be the ratio of the sum of confidence of the rules with the consequent bit of $b_0 = 0$ to the sum of confidences with the consequent bit of $b_0 = 1$. Therefore, in this case, the corruption likelihood of bit#0 will be calculated by the following equation:

$$CR_0 = \frac{\sum Confidence_{(b_0=0)}}{\sum Confidence_{(b_0=1)}} = 0.38 \qquad (1)$$

The corruption likelihood calculated for other bits are shown in Figure 3. Therefore, based on the calculated corruption likelihood, bit number 3 will be identified as the corrupted bit. In the very rare cases where two bits have the same corruption likelihood, a random bit will be recognized as the corrupted one.

IV. Experimental Setup and Results

In this section, we will use the recovery algorithm explained in the previous section to predict the data corruption in the CNNs. Different CNN architectures with different sizes, will be used and different tests will be conducted to check the efficiency of the proposed algorithm.

A. ResNet CNN Architecture and Training

In this paper, we have used ResNet CNN [11] for testing the performance of our recovery algorithm. Two different versions of ResNet, known as ResNet32 and ResNet110 (available in the Keras library) were trained on the CIFAR10 image dataset [28]. The CIFAR10 dataset contains 60,000 RGB 32×32 pixel images labeled into 10 different classes, from which 50,000 images were used for training and the rest were used for testing. Each model was trained to achieve two levels of accuracy by adjusting the number of epochs in the training phase. A dynamic learning rate was set to 0.001 for the epochs in the range of 1 to 80, and 0.1 for the consequent epochs. The aforementioned CNNs consist of two dimensional convolutional layers, batch normalization, ReLU activation, average pooling, and dense in the final layer. More details on the four different CNNs that we have used to conduct our experiments on, are presented in Table II.

Figure 4 demonstrates the frequency of the dominant value (either 0 or 1) for each of the 32 bits, in the binary presentation of the weights of the corresponding CNN model with the accuracy of 0.915 and 0.916. As can be seen, for both models, the dominant value frequency of the 2nd and 4th bits are close to 1.0. For example, the 2nd bit in both CNNs have the value of 0 for more than 99.8% of the weights. Therefore, the naive approach of recognizing the 2nd bit as the corrupted bit, whenever the value of 1 is observed, could give us a high accuracy value of 99.8%. Hence, for the cases of corruption in high frequency bits, the prediction accuracy of the recovery algorithm should exceeds the accuracy of the naive approach, if the evaluation is based on the accuracy metric. Obviously, a study can be conducted to evaluate the performance of the recovery algorithm against the naive approach in terms of other metrics such as sensitivity and specificity.

TABLE II
SPECIFICATIONS OF RESNET MODELS TRAINED ON CIFAR10 IMAGE DATASET.

CNN	#Parameters	#Epochs	Accuracy
ResNet32	470,218	50	0.820
ResNet32	470,218	100	0.915
ResNet110	1,742,716	50	0.839
ResNet110	1,742,716	100	0.916

B. Recovery Algorithm Parameter Tuning

Given the description of our proposed recovery algorithm, it is clear that there are two parameters that influences its performance. The first one is the minimum support level used for finding the frequent patterns in the data (using either apriori or fpgrowth) and the minimum confidence level, which is used for finding and trimming the association rules. It is obvious that lowering the minimum support value would yield inclusion of more frequent patterns and more data would be considered in the extraction of the association rules. However, as a downside, the computational cost of the recovery algorithm would increase drastically.

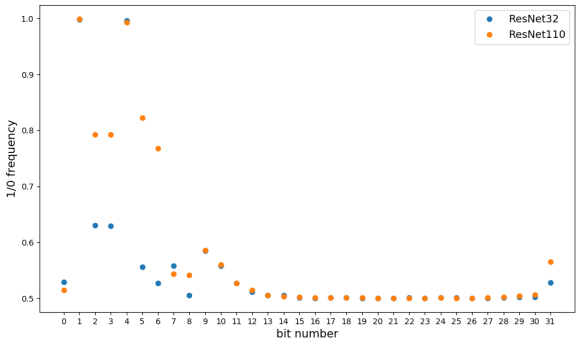


Fig. 4. Frequency of dominant value (either 1 or 0, whichever appears more in the weights) for each of 32 bits in ResNet32 and ResNet110 trained with accuracy of 0.915 and 0.916.

Hence, in an effort to find the best values of those parameters, we have done a series of preliminary experiments with different assigned values for the minimum support level and the minimum confidence value to check the performance of the recovery algorithm. Table III summarizes the conducted tests and represents the success rate of the recovery algorithm applied on the trained ResNet32 with the accuracy of 0.915.

TABLE III
THE SUCCESS RATE OF THE RECOVERY ALGORITHM APPLIED ON THE RESNET32, WITH DIFFERENT MINIMUM SUPPORT LEVEL AND MINIMUM CONFIDENCE VALUES.

Minimum support level	Minimum confidence	Success rate
0.2	0.2	0.60
0.2	0.3	0.84
0.2	0.4	0.80
0.2	0.5	0.64
0.2	0.6	0.60
0.2	0.7	0.60
0.3	0.2	0.64
0.3	0.3	0.80
0.3	0.4	0.76
0.3	0.5	0.40
0.3	0.6	0.76
0.3	0.7	0.80
0.4	0.2	0.52
0.4	0.3	0.56
0.4	0.4	0.56
0.4	0.5	0.36
0.4	0.6	0.64
0.4	0.7	0.52
0.5	0.2	0.40
0.5	0.3	0.52
0.5	0.4	0.08
0.5	0.5	0.36
0.5	0.6	0.40
0.5	0.7	0.48

The conclusion that can be made from Table III is that the lower the value of the minimum support and minimum confidence level, the better the performance of the recovery algorithm. While the best overall performance is seen for the minimum support and minimum confidence of 0.2 and 0.3, respectively, the minimum support level of 0.3 associated with

the minimum confidence of 0.3 shows a relatively good and comparable performance to the best value as well. However, since using the min support level of 0.2, will increase the number of frequent patterns, which consequently increase the time complexity of the recovery algorithm, we chose to proceed with setting both the min support level and the min confidence to 0.3. It should be noted that the success rate of the experiments in Table III is calculated by running 25 runs for each case, and by introducing the corruption of a random bit of a random weight. The number of most significant bits were set to 9. If necessary, similar tests can be performed for other kinds of corruption (presented later in the paper) and different number of most significant bits as well. However, conducting such a combinatorial experiment will be very time consuming and it should be only done for providing a rough estimation on the appropriate values of the parameters, to be later used for different kinds of experiments.

C. Corruption in Random Weight and Random Bit

The first set of experiments we have designed for assessing the recovery algorithm is to introduce a corruption in a single bit of a single weight of the CNN. The number of most significant bits are selected to be 9, 11, and 13 to check its effect on the recovery algorithm. The experiments will be conducted on each of the four available CNNs mentioned in Table II, and for each of the three pre-selected number of most significant bits, which yields a total of 12 experiments. For each experiment, the corruption and recovery algorithm application will occur 100 times. The fraction of instances where the recovery algorithm successfully predicted the corrupted bit will be reported as the accuracy of the algorithm. The results are presented in Table IV.

TABLE IV
ACCURACY OF RECOVERY ALGORITHM ON CORRUPTION INTRODUCED ON A RANDOM BIT AND A RANDOM WEIGHT.

CNN (Accuracy)	msbi = 9	msbi = 11	msbi = 13
ResNet32 (0.820)	0.62	0.54	0.42
ResNet32 (0.915)	0.66	0.48	0.47
ResNet110 (0.839)	0.60	0.61	0.51
ResNet110 (0.916)	0.61	0.57	0.60

Moreover, we have repeated the tests for 30 replicates for each of which the algorithm is tested 100 times. This will help us to find the performance variance of our algorithm. We have performed the replications for the random bit corruption in ResNet32 with the accuracy of 0.915, with different most significant bits of 9, 11, and 13. The mean performance and the variance are shown in Table V.

TABLE V
THE MEAN PERFORMANCE OF THE RECOVERY ALGORITHM OF 30 REPLICATES. A RANDOM BIT IS CORRUPTED FOR RESNET32 WITH THE ACCURACY OF 0.915 AND THE ALGORITHM IS TESTED 100 TIMES.

	msbi = 9	msbi = 11	msbi = 13
Mean recovery rate	0.7210	0.5510	0.4780
Variance	0.0014	0.0054	0.0066

As it can be seen from Table IV, for almost all the cases, the accuracy of the recovery algorithm decreases with the increase in the number of most significant bits. Moreover, it is expected that the required time for finding the frequent pattern increases. Table VI represents the relative time cost of the recovery algorithm with respect to the number of most significant bits used. As can be seen, the time complexity of the recovery algorithm, when 11 and 13 most significant bits are used, increases 35% and 65% compared to the case where 9 bits are included as the most significant bits. This higher time complexity is based on two reasons. First, the volume of the data increases, and second, the number of times the rule mining should be performed increases as well, since the rule mining needs to be done twice for each bit. Therefore, when 13 bits are used, the rule mining will be done for 26 times on data which is approximately 44% larger compared to the 18 times of the rule mining operation when 9 most significant bits are considered.

TABLE VI
AVERAGE RELATIVE ELAPSED TIME FOR PERFORMING THE RECOVERY ALGORITHM ON DIFFERENT CNNS. INCREASING THE NUMBER OF MOST SIGNIFICANT BITS TO 11 AND 13, RESULT IN 35 AND 65% INCREASE IN THE AVERAGE TIME COMPLEXITY, COMPARED TO THE CASE WHERE 9 MOST SIGNIFICANT BITS ARE USED.

CNN (Accuracy)	msbi = 9	msbi = 11	msbi = 13
ResNet110 (0.916)	1.0	1.35	1.65

D. Corruption in Weights with High Frequency

As mentioned earlier, some bits in the CNN weights are dominated with a specific value of either 1 or 0 as illustrated in Figure 4. Therefore, a set of experiments are carried on to test the recovery algorithm on the corruption of those specific bits. As can be seen from Figure 4, bits 1 and 4 have repetition frequency of higher than 99%. Therefore, in this section, we will introduce corruption in the first and fourth bit of a random weight of the CNN and will measure the efficiency of our recovery algorithm.

TABLE VII
ACCURACY OF RECOVERY ALGORITHM ON CORRUPTION INTRODUCED ON BIT #1 - DIFFERENT NUMBER OF MOST SIGNIFICANT BITS ARE USED IN RULE MINING ALGORITHM.

CNN (Accuracy)	msbi = 9	msbi = 11	msbi = 13
ResNet32 (0.820)	0.99	0.99	1.00
ResNet32 (0.915)	1.00	1.00	1.00
ResNet110 (0.839)	0.98	1.00	1.00
ResNet110 (0.916)	1.00	1.00	1.00

Based on the presented results, for the case where the corruption is introduced on the bit#1 of a random weight, the recovery algorithm shows very good performance. Out of 12 different types of experiments presented in Table VII, the recovery algorithm yielded a 100% accuracy, and two of them with the accuracy of 99%. It seems that the dimension of the CNN does not affect the accuracy of the recovery algorithm in this specific case and the recovery algorithm

TABLE VIII
ACCURACY OF RECOVERY ALGORITHM ON CORRUPTION INTRODUCED ON BIT #4 - DIFFERENT NUMBER OF MOST SIGNIFICANT BITS ARE USED IN RULE MINING ALGORITHM.

CNN (Accuracy)	msbi = 9	msbi = 11	msbi = 13
ResNet32 (0.820)	1.0	1.0	0.98
ResNet32 (0.915)	1.0	0.99	1.0
ResNet110 (0.839)	0.8	0.84	0.74
ResNet110 (0.916)	0.71	0.82	0.76

works well for both ResNet32 and ResNet110. Moreover, repeating the test for 30 replicates on ResNet32 (0.915), the mean performance of the recovery algorithm was found to be 0.994 and 0.996 for the cases of the corruption at bit#1 and bit#4, with low variance of 0.00004 and 0.00003, respectively. The low variance in the performance of the algorithm confirm its effectiveness in the good prediction ability of the corrupted bit. However, looking at Table VIII, where the performance of the recovery algorithm on the weights with corruption on bit#4 is presented, the performance is sub-par for the ResNet110 with the maximum accuracy of 84%. Therefore, since bit#4 has a value of 1 for more than 99% of the time, whenever its value is observed to be 0, identifying it as a corrupted bit without any further analysis, could result in better performance compared to the recovery algorithm. However, for ResNet32, the performance is acceptable compared with the naive approach.

E. Corruption in Specific Types of Weights

In ResNet, most of the weights are either 2-dimensional convolutional coefficients (Conv2D) or batch normalization coefficients. In this set of experiments, we will isolate the Conv2D and batch normalization coefficients and will only use these coefficients for mining the association rules and checking the performance of our recovery algorithm. The Conv2D weights corresponds to approximately 98.9% and 99.9% of all weights in ResNet32 and ResNet110, respectively. Batch normalization coefficients constitute a much smaller fraction of the weights including 0.01% for both ResNet32 and ResNet110 CNNs. Moreover, the quantitative distribution of all the weights, Conv2D, and batch normalization weights can be seen in Figures 5.

Table IX represents the accuracy of the recovery algorithm on the corruptions introduced on the Conv2D and batch normalization weights. The first 9 bits of the binary representation were used for constructing the recovery algorithm.

TABLE IX
ACCURACY OF RECOVERY ALGORITHM ON CORRUPTIONS INTRODUCED ON CONV2D AND BATCH NORMALIZATION WEIGHTS.

CNN (Accuracy)	Conv2D	Batch normalization
ResNet32 (0.820)	0.64	0.66
ResNet32 (0.915)	0.68	0.59
ResNet110 (0.839)	0.64	0.62
ResNet110 (0.916)	0.62	0.61

Comparing the results presented in Table IX with those presented in Table IV, no specific pattern can be seen since for some cases like ResNet32 (0.915) there is an improvement in the recovery algorithm for Conv2D weights, while for the batch normalization the algorithm performance drops down. Overall, no notable improvement in performance can be seen when the weights are isolated.

V. CONCLUSION AND FUTURE WORK

A recovery algorithm was presented for recovering the data corruption in the case of SEU. The ResNet32 and ResNet110 CNN were trained on the CIFAR10 dataset, with two different accuracy levels for each of them. The number of parameters involved with ResNet110 is 3.7 times greater than that of ResNet110. The Apriori and FP-Growth data mininig algorithms were used for finding the frequent patterns and the associated confidence level with the obtained rules. The confidence levels were used later to define a probabilistic measure for identifying the corruption possibility.

The pre-trained parameters are stored in the format of IEEE-754, which uses a 32 bit binary. Therefore, the bits closer to the far-right side of the binary representation will have a marginal and insignificant effect on the final value of that parameter. Hence, we have reduced the number of bits in this research study to 9, 11, and 13 bits to investigate its effect on our recovery algorithm. The recovery algorithm was noticed to drop when the number of most significant bits are increased. In the case of using 9 significant bits, the recovery rate was observed to be in the range of 61 to 66% when the errors were introduced for a random bit. Moreover, in some bits, more than 99% of all the parameters are showing to have one specific value. The performance of the algorithm was assessed in the case of corruption for those bits. For some bits, the recovery rate was noticed to be in the range of 98 to 100%, however, for some other frequent bits and specially for ResNet110 the performance declined as low as 74%.

In this paper, the assumption of SEU was made which limits the corruption level at only one bit. Research on developing algorithms for recovery of the corrupted weights with multiple bits is also of interest. While this work was focused on the recovery of corruption, rather than detection with the assumption that the corrupted weight is already identified, future work will develop algorithms for simultaneous detection and recovery.

REFERENCES

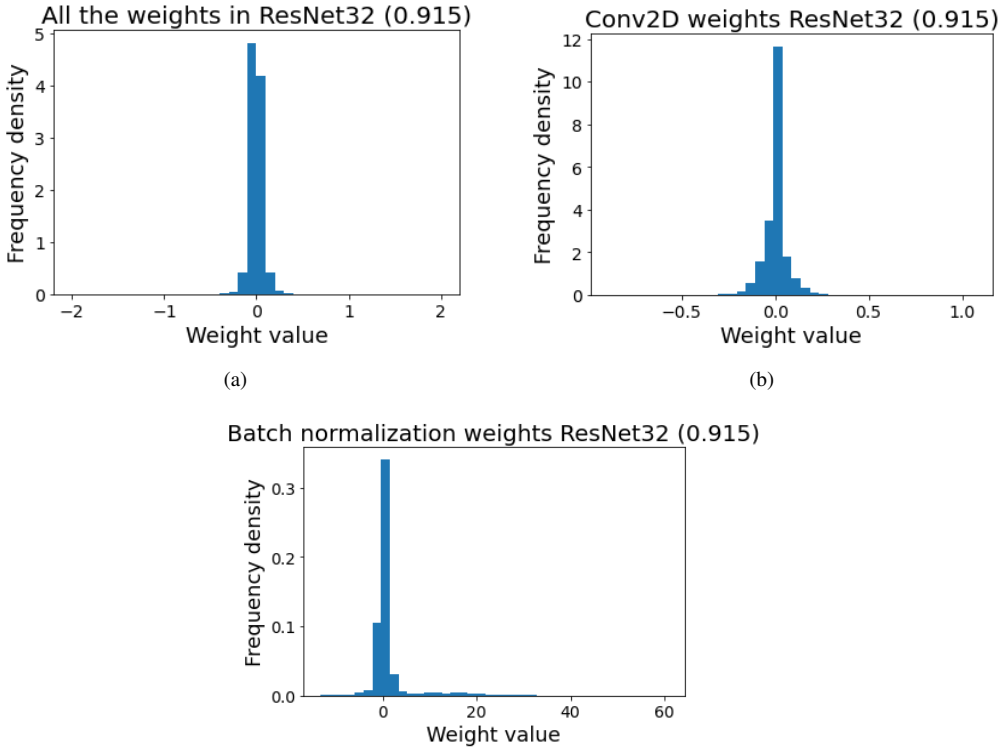
[1] X. Guo, X. Liu, E. Zhu, J. Yin, Deep clustering with convolutional autoencoders, in: International conference on neural information processing, Springer, 2017, pp. 373–382.

[2] S.-H. Liao, C.-H. Wen, Artificial neural networks classification and clustering of methodologies and applications–literature analysis from 1995 to 2005, Expert Systems with Applications 32 (1) (2007) 1–11.

[3] C. Wang, M. Cheng, F. Sohel, M. Bennamoun, J. Li, Normalnet: A voxel-based cnn for 3d object classification and retrieval, Neurocomputing 323 (2019) 139–147.

[4] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

[5] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.



(a) (b) (c)

Fig. 5. Histogram for quantitative distribution of (**a**) all weights, (**b**) Conv2D weights, and (**c**) batch normalization weights in ResNet32 (0.915)

[6] K. He, G. Gkioxari, P. Dollár, R. Girshick, Mask r-cnn, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 2961–2969.

[7] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[8] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014).

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[10] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[11] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[12] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167 (2015).

[13] Z. Wu, C. Shen, A. Van Den Hengel, Wider or deeper: Revisiting the resnet model for visual recognition, Pattern Recognition 90 (2019) 119–133.

[14] H. Meng, N. Pears, C. Bailey, A human action recognition system for embedded computer vision application, in: 2007 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2007, pp. 1–6.

[15] M. Moshayedi, B. H. Robinson, System and method for preventing data corruption in solid-state memory devices after a power failure, uS Patent 7,107,480 (Sep. 12 2006).

[16] T. C. May, M. H. Woods, A new physical mechanism for soft errors in dynamic memories, in: 16th International Reliability Physics Symposium, IEEE, 1978, pp. 33–40.

[17] E. Normand, Single event upset at ground level, IEEE transactions on Nuclear Science 43 (6) (1996) 2742–2750.

[18] R. Baumann, T. Hossain, S. Murata, H. Kitagawa, Boron compounds as a dominant source of alpha particles in semiconductor devices, in: Proceedings of 1995 IEEE International Reliability Physics Symposium, IEEE, 1995, pp. 297–302.

[19] M. Riera, R. Canal, J. Abella, A. Gonzalez, A detailed methodology to compute soft error rates in advanced technologies, in: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2016, pp. 217–222.

[20] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, R. Brightwell, Detection and correction of silent data corruption for large-scale high-performance computing, in: SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, 2012, pp. 1–12.

[21] K. M. Zick, C.-C. Yu, J. P. Walters, M. French, Silent data corruption and embedded processing with nasa's spacecube, IEEE Embedded Systems Letters 4 (2) (2012) 33–36.

[22] B. Charyyev, A. Alhussen, H. Sapkota, E. Pouyoul, M. H. Gunes, E. Arslan, Towards securing data transfers against silent data corruption., in: CCGRID, 2019, pp. 262–271.

[23] X. Ni, L. V. Kale, Flipback: automatic targeted protection against silent data corruption, in: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2016, pp. 335–346.

[24] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, F. Cappello, Lightweight silent data corruption detection based on runtime data analysis for hpc applications, in: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, 2015, pp. 275–278.

[25] R. Agrawal, R. Srikant, et al., Fast algorithms for mining association rules, in: Proc. 20th int. conf. very large data bases, VLDB, Vol. 1215, 1994, pp. 487–499.

[26] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, ACM sigmod record 29 (2) (2000) 1–12.

[27] IEEE standard for floating-point arithmetic, IEEE Std 754-2008 (2008) 1–70.

[28] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).