DEVELOPMENT OF AN ALGORITHM FOR DETECTION AND RECOVERY OF CORRUPTION IN CONVOLUTIONAL NEURAL NETWORKS DATA STORAGE

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Mohammadreza Ramzanpour

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

March 2021

Fargo, North Dakota

North Dakota State University
Graduate School

Title

DEVELOPMENT OF AN ALGORITHM FOR DETECTION AND RECOVERY OF CORRUPTION IN CONVOLUTIONAL NEURAL NETWORKS DATA STORAGE

By

Mohammadreza Ramzanpour

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Saeed Salem

Dr. Di Wu

Approved:

03/21/2021	Dr. Simone Ludwig
Date	Department Chair

ABSTRACT

Computer vision based applications are commonly utilized in embedded systems. The demand for higher accuracy leads to increased complexity of convolutional neural networks (CNN) and hence, larger storage requirement for saving pre-trained networks. Even the smallest data corruption in the storage units of the embedded systems can result in drastic failures due to the propagation of the errors. This thesis proposes a new algorithm for the recovery of the data in case of single event upset (SEU) error. An association rule mining based algorithm is used to find the probability of corruption in each of the bits. The recovery algorithm is tested on four different trained ResNet and the best recovery rate of 66% was found in the most complex scenario. However, for the special cases of corruption in the frequently repeated bits, the recovery rate was found to be perfect with 100% recovery rate.

ACKNOWLEDGEMENTS

Firstly, I express sincere gratitude to my advisor, Professor Simone Ludwig, for her guidance, great helps, understanding, motivation, and immense knowledge.

I thank Professor Saeed Salem, and Professor Di Wu for their time and for agreeing to serve as committee members.

Special thanks to Professor Ghodrat Karami for his support of the continuation of my endeavors in this way.

I want to deliver my gratitude to Professor Vasant Ubhaya and Professor Joseph Latimer for all their encouragements and for being such excellent teachers.

I am thankful to Dr. Rahil Ashtarimahini for her great helps and companionship in this path and to my many friends and colleagues including Dr. Mohammad Hosseini-Farid, Dr. Mojtaba Ahmadi, and Dr. Jamileh Shojaeiarani.

The last but not the least, I want to express my gratitude to my dear parents, my dear sisters Dr. Maryam, Dr. Melika, Neda and my lovely nephew Amirali which their support and friendship always mean a lot to me.

DEDICATION

I am honored to dedicate this thesis to my dear family.

To my beloved father, who my greatest wish in life is to be a man as great as he is. To my dear mother who is a symbol of unconditional love to me.

My heart beats for you!

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGEMENTS iv

DEDICATION v

LIST OF TABLES vii

LIST OF FIGURES viii

1. INTRODUCTION 1

2. RELATED WORKS 8

3. METHODOLOGY 12

4. EXPERIMENTAL SETUP AND RESULTS 17

4.1. ResNet CNN Architecture and Training 17

4.2. Recovery Algorithm Parameter Tuning 18

4.3. Corruption in Random Weight and Random Bit 20

4.4. Corruption in Weights with High Frequency 22

4.5. Corruption in Specific Types of Weights 23

5. CONCLUSION AND FUTURE WORKS 26

REFERENCES 27

LIST OF TABLES

Table Page

1.1. Some of the famous CNN architectures with the number of parameters involved. The number of parameters for modern architectures vary in the range of 4 to more than 100 million parameters. 2

4.1. Specifications of ResNet models trained on CIFAR10 image dataset. 18

4.2. The success rate of the recovery algorithm applied on the ResNet32, with different minimum support level and minimum confidence values. 19

4.3. The relative time complexity of the Apriori and FP-growth algorithms applied in our recovery algorithm. All the reported numbers are relative to the case of Apriori with most significant bits of 9. Tests were done on ResNet32 (0.915) with 20 repetitions. . . . 20

4.4. Accuracy of the recovery algorithm on corruption introduced on a random bit of a random weight. 21

4.5. 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. 21

4.6. Average relative elapsed time for performing the recovery algorithm on ResNet110 (0.916) using Apriori algorithm. 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. 22

4.7. Accuracy of recovery algorithm on corruption introduced on bit #1 - different number of most significant bits are used in rule mining algorithm. 22

4.8. Accuracy of recovery algorithm on corruption introduced on bit #4 - different number of most significant bits are used in rule mining algorithm. 23

4.9. Accuracy of the recovery algorithm on corruptions introduced on Conv2D and batch normalization weights. 24

LIST OF FIGURES

Figure Page

1.1. Illustration of residual learning in one layer. The data from one layer is directly transferred to the next layer. This approach is beneficial for overcoming the problem of vanishing gradients [1]. 3

1.2. The plain CNN architecture with 34 layers that can be used for the image classification task [1]. 4

1.3. The ResNet34 architecture. 34 layers are present in this architecture with skip inter-layer connections [1]. 5

3.1. A sample corrupted weight with 9 most significant bits ($n = 9$) with the corruption at the bit#3. 13

3.2. A sample rule mining for bit#0 as the consequent bit. 16

3.3. Calculated corruption likelihood for bits of weight shown in Fig. 3.1. 16

4.1. 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. . . 18

4.2. Histogram for quantitative distribution of all the weights in trained ResNet32 with accuracy of 0.915. 24

4.3. Histogram for quantitative distribution of Conv2D weights in trained ResNet32 with accuracy of 0.915. 25

4.4. Histogram for quantitative distribution of batch normalization weights in trained ResNet32 with accuracy of 0.915. 25

1. 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 subdivided 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 [2, 3]. 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 known as deep learning (DL) methods, which refers to their layered structure.

Convolutional neural network (CNN) is a DL approach, which is commonly used in computer vision applications such as image classification [4, 5], object detection [6–8], semantic segmentation [9–11], and instance segmentation [12–14]. CNN is based on a mathematical operation named as convolution, which is accomplished by the use of kernels (also known as filters) [15]. 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, usually, CNNs are commonly accompanied by several fully connected layers for performance improvement [16]. In many of the CNN architectures, these layers account for a 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 1.1. As it can be seen from Table 1.1, 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 [17]. 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, new methodologies have 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

Table 1.1. Some of the famous CNN architectures with the number of parameters involved. The number of parameters for modern architectures vary in the range of 4 to more than 100 million parameters.

Architecture name	Number of parameters
AlexNet [18]	60M
VGG [19]	138M
GooLeNet [20]	4M
DenseNet [21]	15M-25M

the network causes an issue in the training phase known as the vanishing gradient problem [22]. Even the use of augmented layers like inception networks [20], which combines multiple kernels and pooling layers into one, may suffer from the vanishing gradient. The ResNet architecture originally introduced by He et al. [1] 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 [23]. 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 [24]. The skipping layers and transfer of data from one layer to next layers can be seen in Figure 1.1 [1]. The ResNet architecture uses many of these kind of layers and is used as the backbone in many of the developed architectures for different applications such as semantic segmentation. For example, out of four available pre-trained deep learning models in the torchvision [25] package of PyTorch [26], two of them use the ResNet backbone architecture including ResNet50 and ResNet101. However, to adapt them for different sizes of input images, they are modified to be fully convolutional and hence, max pooling layers are not used. Figure 1.2 and Figure 1.3 show plain CNN and ResNet both with 34 layers.

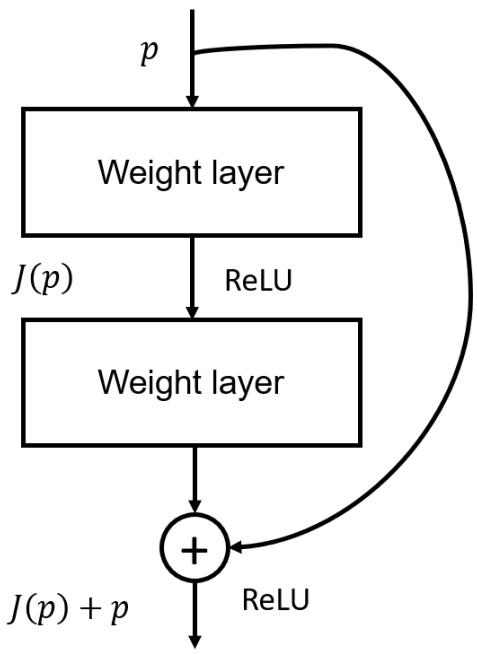

Figure 1.1. Illustration of residual learning in one layer. The data from one layer is directly transferred to the next layer. This approach is beneficial for overcoming the problem of vanishing gradients [1].

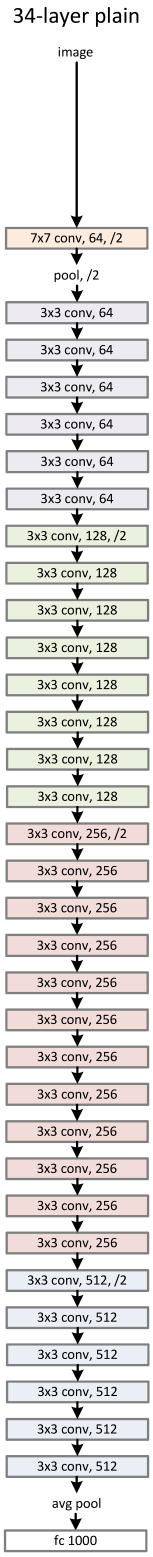

Figure 1.2. The plain CNN architecture with 34 layers that can be used for the image classification task [1].

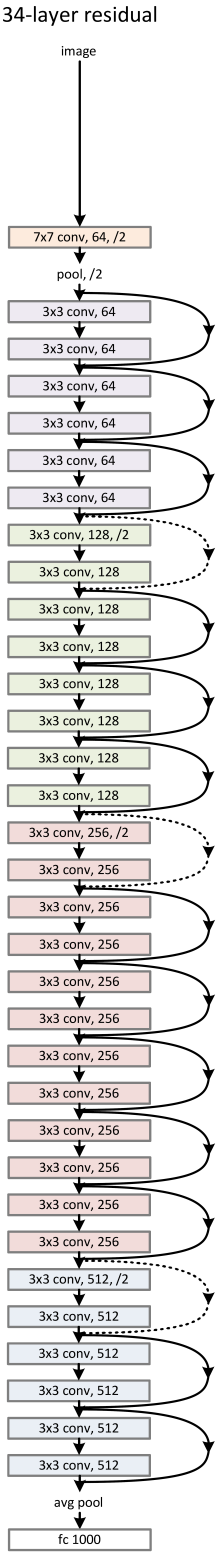

Figure 1.3. The ResNet34 architecture. 34 layers are present in this architecture with skip inter-layer connections [1].

Embedded system is a term referring to a category of computing based devices that can be a combination of hardware and software in the context of electrical or even mechanical devices [27]. 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 [28] such as autonomous cars [29], smart home security [30], and facial recognition gadgets [31] 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 embedded systems [32], 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 [33], high energy neutron from cosmic radiations [34], and low energy cosmic neutron reaction with Boron in integrated circuits (IC) [35] can be mentioned. These soft errors cannot be detected by the operating system unless it results in some data structure corruption or invalidating the integrity of the system. Soft errors can be divided into two types of single event upset (SEU) and burst errors [36]. 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) [37], while in the burst error case, multiple bits (usually consecutive ones) get corrupted and their values change [38]. Using parity and/or checksum bits can be useful in detection of SEU errors, 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, 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) [39].

In this thesis, an algorithm is developed for detecting the corrupted bit in the case of SEU. Association rule mining 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 subsequently, the bit with the highest probability of corruption will be determined as the corrupted bit.

This thesis is derived from the following publication [40]:

2. RELATED WORKS

Many attempts have been made to make and improve hardware for better resiliency against data corruption. However, this problem still cannot be avoided completely [41] and hence, the need for software-based detection and recovery of the corrupted data remains a necessity.

Approaches known as redundancy techniques aim to increase the reliability of the computing system by using data backup, which increases the storage requirement. Avizienis [42] discusses three redundancy categories including hardware, software, and time redundancy that are all used against hardware failure. Hardware redundancy refers to the components and circuits designed to prevent such errors and are sub-categorized into two approaches of masking and recovery. For example, masking uses replication of the components in the system and whenever one component fails, it will be masked by the other ones. Component quadding and triple modular redundancy (TMR) are among the techniques that belong to this approach [42]. Software redundancy deals with the aftermath of the fault occurrence where an emergency program will be executed to mitigate the error. This program should be frequently executed to check all the logic circuits for the existence of such errors. Finally, repetition of a program (or some section of it) to provide a comparative analysis on the results is known as time redundancy [43].

Nocolaidis et al. [44] discuss different error correcting codes (ECC). They refer to the parity bit method as one of the simplest approaches for mitigation of soft errors, however, they assert that while these approaches are capable of detecting the occurrence of bit corruptions, they are not capable of resolving the errors of this kind and they must be accompanied with system level approaches for resolving the error. Nicolaidis mentions about the overhead time complexity of the replication and TMR approaches (100% and 200%) and further introduces reusing scan-path resources as a way to overcome this issue [45].

LaBel et al. [46] studied the effect of cosmic radiation on the performance of microelectronic technologies in space. They introduce several approaches for mitigation of soft errors from which Hamming code [47], Reed-Solomon (RS) code [48], convolutional encoding [49], and overlaying protocol can be mentioned. While the hamming code can detect the corruption of up to two bits, it can correct one of the flipped bits only if that specific bit is somehow known. They suggest the

hamming code to be used for the systems that have low probability of multiple bit corruption. The RS code can find and correct multiple consecutive corrupted bits in a single data structure. While the encoding of data with RS code is relatively simple, the decoding process (detection of error) is computationally expensive. The RS code encodes groups of bits instead of one bit at a time approach used in Hamming code. The RS code constructs a $k-1$ degree polynomial for sending k bits and checks if the received bits can be correctly evaluated from that polynomial. By creating a redundancy and sending s additional nodes, it would be able to detect the error in case that less than s bits are corrupted.

Field programmable gate array (FGPA) systems are integrated circuits (IC) that can be further designed by customers beyond the manufacturer settings. Static random access memory (SRAM) based FGPAs are prone to SEU and the induced SEU can appear in both internal memory cells and FGPA configuration memory. Legat et al. [50], proposed a self repairable system for FGPA based systems. This self repairable system occupies a small fraction of the FGPA memory and the rest can be used for the target application. The system detects and repairs the soft errors while the target application is running. This self-repairable system consists of an internal recovery mechanism program and an external watchdog timer. In their system, control logic component, which has finite state machine (FSM), frame address counter, and error location logic is in charge of detecting the errors. The FSM is responsible for the internal recovery mechanism and the external watchdog timer is in charge of the external recovery.

Due to the overhead caused by error detection algorithms, the energy consumption increases, and therefore, a trade-off between energy and fault tolerant features should be in effect. This was the focus of a study conducted by Ejlali et al. [51]. While dynamic voltage scaling (DVS) is used to decrease the energy consumption, the time redundancy fault detection systems depletes the resources, and this is where a balance should be made. They proposed the use of information redundancy alongside time redundancy to achieve both targets of reducing energy usage and increasing fault tolerance. The authors have compared two different fault-aware systems namely conventional R systems and their own proposed RI system. The conventional system use pure rollback recovery mechanism, which is encompassed by time redundancy approaches, i.e., whenever transient faults such as SEU happens, a recovery execution of the same task is needed. Their proposed approach takes advantage of both rollback recovery and information redundancy [52] that is the duplication

of data and storing mirror copies in different locations. Whenever an SEU occurs, the yielded error would be corrected by the use of additional hardware. By testing this technique, they were able to reduce the energy consumption up to 44%.

While the above mentioned research focused on employing both hardware and software for mitigating soft errors, there are some other studies that have only used software approaches for resolving such an issue.

Fiala et al. [53] 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. [54] developed a method for detecting SDC 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. [55] 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. [56] proposed a software based approach named FlipBack, for guarding the applications against the SDC. 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. [57] 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 subsequent 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 worthy 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 (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 thesis and is outlined in the following sections.

3. METHODOLOGY

The Apriori algorithm introduced by Agrawal and Srikant [58] is one of the most common used frequent patterns finding algorithms. The Apriori algorithm is based on the fact that if an itemset consisting of n items is frequent, then each subset of that itemset with the size varying in range of 0 to $n-1$ must be a frequent set as well. In an iterative incremental fashion, this algorithm first finds the frequent itemsets with a single item. In the subsequent iterations, it merges the previously found frequent itemsets 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 scanned each time when an itemset is checked to be frequent or not. Moreover, many spurious candidate frequent itemsets may be generated, which could cause bottlenecks in terms of memory usage.

The frequent pattern growth (FP-growth) algorithm introduced by Han et al. [59] 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 frequent pattern-tree (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 FP-tree for storing the frequent patterns and prefix analysis leads to a more efficient algorithm in term of memory consumption [60].

In this thesis, the Apriori algorithm is used for the cases involved with ResNet32 and the FP-growth algorithm for the cases where ResNet110 were studied, 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 frequent patterns finding 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, different aspects of this algorithm will be delineated. First, 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 [61]. At this stage, there is a $N \times 32$ matrix where N refers to the total number of weights of the model. The n first bits of the binary representation can be selected to serve 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 lack of its presence for the cases with value of 0.

The corruption prediction is based on the assumption that the corruption happens at only one bit of a single weight. A sample corrupted weight is shown in Figure 3.1 with $n = 9$, i.e., 9 significant bits. In this example, an assumption was made that corruption has taken place in the bit#3.

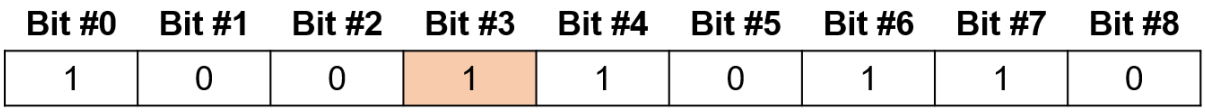

Figure 3.1. A sample corrupted weight with 9 most significant bits ($n = 9$) with the corruption at the bit#3.

For each bit of the weight in study (the corrupted bit), 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 is or was originally valued as 1 and second time for the value of 0. The bit being examined will be considered as the consequent bit and all the other bits will be used as the antecedent bit [1] 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 1 and 0. 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 1, 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 thesis, the minimum support threshold of 0.3 was used for finding the frequent patterns and the confidence threshold value of 0.3 was used for mining the rules as well. These values were fine-tuned based on our conducted experiments to achieve best results. For finding the frequent patterns, the Apriori [58] or FP-growth [59] algorithms was used. Apriori

[1] Here, the term bit serves equivalently as that of the item in the rule mining common terminology

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

was used for the smaller CNNs and the FP-growth was used for the larger ones with more weights, since Apriori requires larger amount of memory. However, the speed of the Apriori was found to be higher compared to the FP-growth algorithm. It is worth mentioning that the number of most significant bits can be adjusted and even all the 32 bits could be used in the recovery algorithm. However, moving toward the rightmost bits, the corruption would have more negligible effects on the value of the corrupted weight. Moreover, the time complexity and computational cost of the frequent pattern mining will increase dramatically as the number of most significant bits increases. The "mlxtend" package [62] in Python was used for rule mining through Apriori and FP-growth algorithms.

The confidence of each mined rule will be calculated as the following given in [63]. With the assumption that two itemsets such as X and Y are found, forming the rule $X \rightarrow Y$, the confidence of this rule can be calculated by the following equation:

$$c(X, Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{3.1}$$

where $c(X, Y)$ is the confidence of rule $X \rightarrow Y$, and $\sigma(X)$ denotes the support count for itemset X. The support count is defined as the following:

$$\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in T\}| \tag{3.2}$$

where operator $|.|$ returns the number of elements in the itemset passed as an operand into it. t_i denotes the i^{th} transaction from the transactions database $T = \{t_1, t_2, ..., t_n\}$. In plain language, the confidence of rule $X \rightarrow Y$ determines the frequency of item Y appearing in the transaction containing X. Support count determines the number of times that the given rule is applicable to the transactions database [63].

To further clarify, consider the example shown in Figure 3.1 where the bit#3 is corrupted but presumably this is not known to us. For the start, bit#0 will be considered for rule mining to determine its probability of being corrupted. In the first step, association of bit#0 with other values must be checked by finding the frequent patterns with setting this bit as the consequent bit. 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 3.2.

Since, the original value of bit#0 (Figure 3.1) 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 of the rules 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:

$$CR_0 = \frac{\sum Confidence_{(b_0=0)}}{\sum Confidence_{(b_0=1)}} = 0.38 \tag{3.3}$$

where CR_0 represent the corruption likelihood value for bit#0 and b_0 denotes the value of bit#0 (1 or 0). The corruption likelihood calculated for other bits are shown in Figure 3.3. Thereafter,

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

Figure 3.2. 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

Figure 3.3. Calculated corruption likelihood for bits of weight shown in Fig. 3.1.

based on the calculated corruption likelihood, bit#3 will be identified as the corrupted bit. In the rare cases where two bits have equal corruption likelihood value, a random choice from the bits in question will be used to determine the corrupted one. Thousands of tests were performed for this dissertation, but this state was not observed for any of the tests.

4. EXPERIMENTAL SETUP AND RESULTS

In this section, the explained recovery algorithm will be used to predict the corrupted bit 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.

4.1. ResNet CNN Architecture and Training

In this thesis, ResNet CNN [1] is used for testing the performance of the proposed recovery algorithm. Two different versions of ResNet, known as ResNet32 and ResNet110 (available in the Keras library) were trained on the CIFAR10 image dataset [64]. 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 subsequent epochs. The aforementioned CNNs consist of two-dimensional convolution (Conv2D), batch normalization, rectified linear unit (ReLU) activation, average pooling, and dense (flatten) components with the last one appearing in the final layer. More details on the four trained CNNs that were used to conduct different experiments on are presented in Table 4.1.

Figure 4.1 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 2^{nd} and 4^{th} bits are close to 1.0. For example, the 2^{nd} bit in both CNNs have the value of 0 for more than 99.8% of the weights. Therefore, the naive approach of recognizing the 2^{nd} 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 these 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 separate 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 4.1. 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

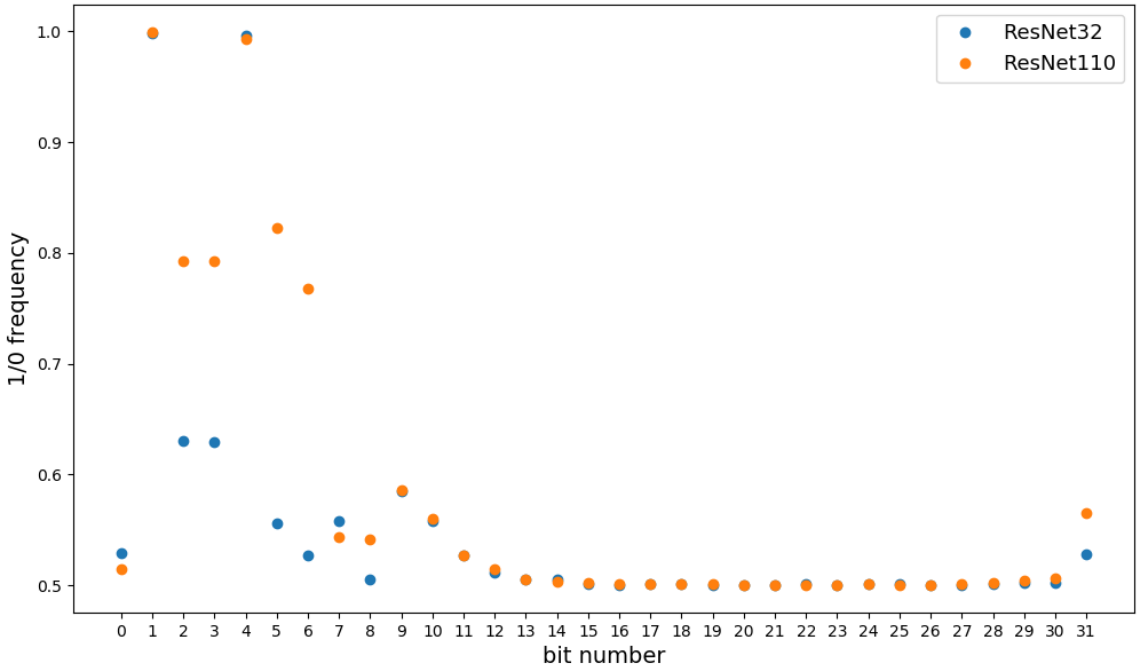

Figure 4.1. Frequency of dominant value (either 1 or 0, whichever appears more in the weights) for each of 32 bits in ResNet32 and ResNet110 trained with accuracy of 0.915 and 0.916.

4.2. Recovery Algorithm Parameter Tuning

Given the description of the 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 FP-growth) 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.

Table 4.2. 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

Hence, in an effort to find the best values of those parameters, a series of preliminary experiments with different assigned values for the minimum support level and the minimum confidence value were conducted to check the performance of the recovery algorithm. Table 4.2 summarizes the conducted tests and represents the success rate of the recovery algorithm applied on the trained ResNet32 with the accuracy of 0.915. The conclusion that can be made from Table 4.2 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 minimum support level of 0.2, will increase the number of frequent patterns, which consequently increase the time complexity of the recovery algorithm, it

was chosen to proceed with setting both the minimum support level and the minimum confidence as 0.3. It should be noted that the success rate of the experiments in Table 4.2 is calculated by running 25 runs for each case, and by introducing the corruption on 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 thesis) and different number of most significant bits as well. However, conducting such combinatorial experiments will be very time consuming and it should be only done for providing a rough estimation on the appropriate values of the threshold parameters, to be later used for different kinds of experiments.

4.3. Corruption in Random Weight and Random Bit

The first set of experiments designed for assessing the recovery algorithm is to introduce a corruption in a single bit of a random 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. As mentioned before, the Apriori algorithm implementation in "mlxtend" package was found to be faster compared to the FP-growth algorithm used in the context of the developed recovery algorithm. Here, for comparison purposes, this experiment is performed for 20 times and reported the average relative time (with the Apriori time complexity on ResNet32 with 9 most significant bits to be considered as 1 unit of time) spent on those two algorithms, as shown in Table 4.3. As can be seen from the table, the time it takes for the FP-growth algorithm to find the frequent patterns in the recovery algorithm is nearly 4 times greater than that of the Apriori algorithm when 9 bits are considered as the most significant.

The random bit corruption experiment was conducted on each of the four available CNNs mentioned in Table 4.1, and for each of the three pre-selected number of most significant bits yielding total of 12 experiments. For each experiment, the corruption and recovery algorithm application was performed for 100 times. The fraction of instances where the recovery algorithm successfully

Table 4.3. The relative time complexity of the Apriori and FP-growth algorithms applied in our recovery algorithm. All the reported numbers are relative to the case of Apriori with most significant bits of 9. Tests were done on ResNet32 (0.915) with 20 repetitions.

Algorithm	msbi=9	msbi=11	msbi=13
Apriori	1	2.18	4.07
FP-growth	4.01	5.51	7.23

Table 4.4. Accuracy of the recovery algorithm on corruption introduced on a random bit of 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

predicted the corrupted bit was reported as the accuracy of the algorithm. The results are presented in Table 4.4.

Moreover, the tests are repeated for 30 replicates in each of which, the algorithm is tested 100 times. This will help to find the variance of the recovery algorithm performance. The replications were performed 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 4.5.

Table 4.5. 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 4.4, for almost all the cases, the accuracy of the recovery algorithm decreases as the number of most significant bits increases. Moreover, it is expected that the required time for finding the frequent patterns increases. Table 4.6 represents the relative time cost of the recovery algorithm with respect to the number of most significant bits used for the ResNet110. 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 4.6. Average relative elapsed time for performing the recovery algorithm on ResNet110 (0.916) using Apriori algorithm. 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

4.4. 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 4.1. Therefore, a set of experiments were carried on to test the recovery algorithm on the corruptions happening on those specific bits. As can be seen from Figure 4.1, bits 1 and 4 have repetition frequency of higher than 99%. Therefore, in this section, the SEU is introduced in the first and fourth bit of a random weight of the CNN, and the mean efficiency of the recovery algorithm (for 100 repeats) is calculated as demonstrated in Tables 4.7 and 4.8.

Table 4.7. 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 4.7, the recovery algorithm yielded 100% accuracy, and two of them provided the accuracy of 99%. It seems that the size of the CNN does not affect the accuracy of the recovery algorithm in this specific case and the recovery algorithm 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 performance of the algorithm confirm its effectiveness and ability in good prediction of the corrupted bit. However, looking at Table 4.8, where the performance of the recovery algorithm

Table 4.8. 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

on the weights with corruption on bit#4 is presented, the performance of the recovery algorithm on the ResNet110 was seen to be yielding 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 to the naive approach.

4.5. Corruption in Specific Types of Weights

In ResNet architecture, most of the weights are either Conv2D or batch normalization coefficients. In another set of experiments presented here, the Conv2D and batch normalization coefficients are isolated and only these coefficients are used for mining the association rules and checking the performance of the recovery algorithm. The Conv2D weights correspond 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 for both ResNet32 and ResNet110. The quantitative distribution of all the weights, Conv2D, and batch normalization weights in ResNet32 can be seen in Figures 4.2, 4.3, and 4.4.

Table 4.9 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. It should be noted that for each weight type, the rule mining was done using only the parameters of that type by excluding other types of parameters.

Table 4.9. Accuracy of the 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 4.9 with those presented in Table 4.4, 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, recovery performance drops down. Therefore, no notable improvement in performance is observed when the weights are isolated.

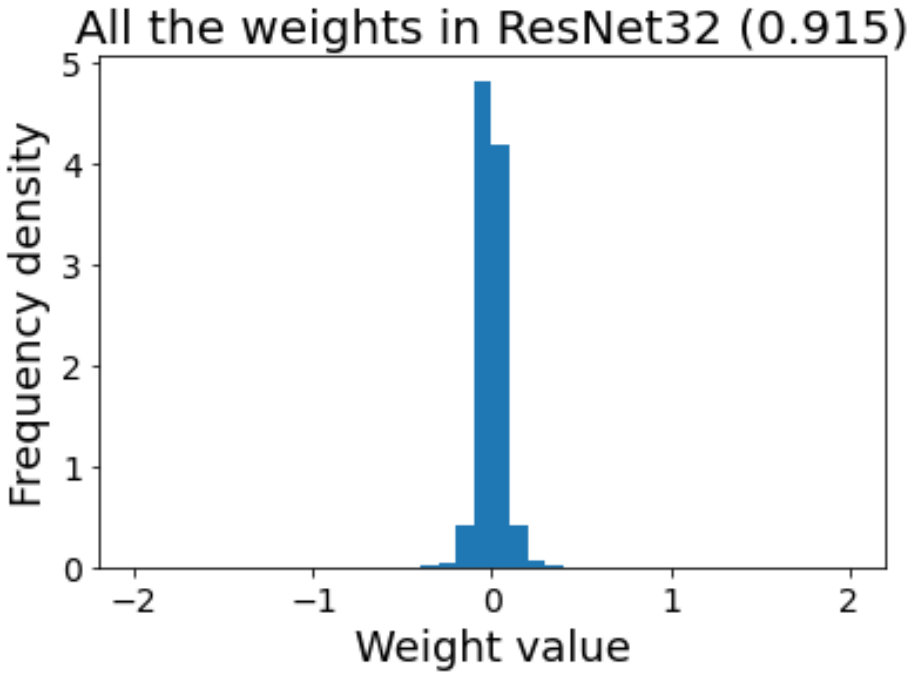

Figure 4.2. Histogram for quantitative distribution of all the weights in trained ResNet32 with accuracy of 0.915.

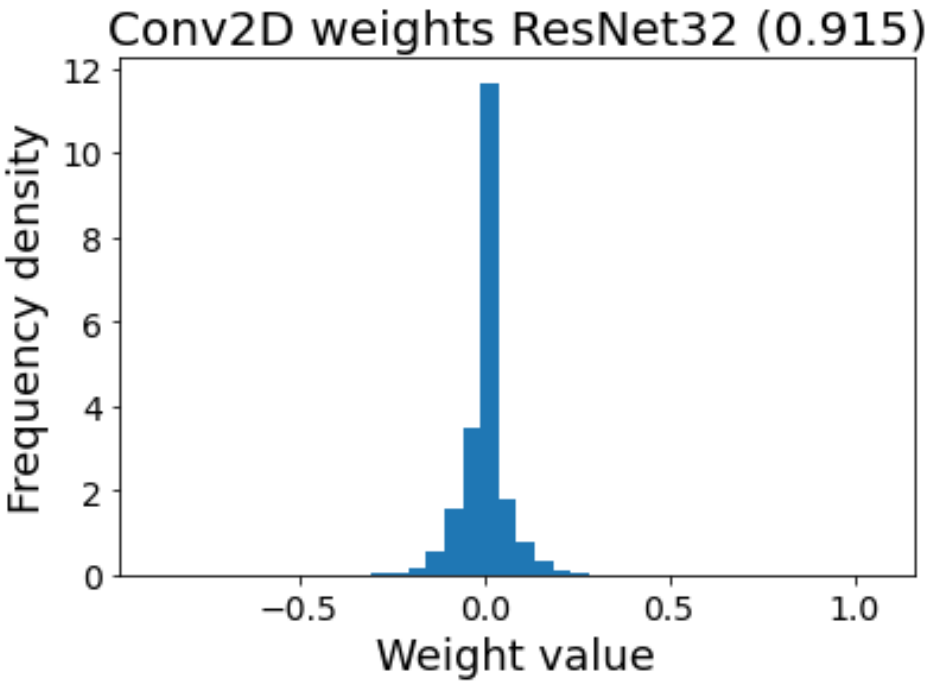

Figure 4.3. Histogram for quantitative distribution of Conv2D weights in trained ResNet32 with accuracy of 0.915.

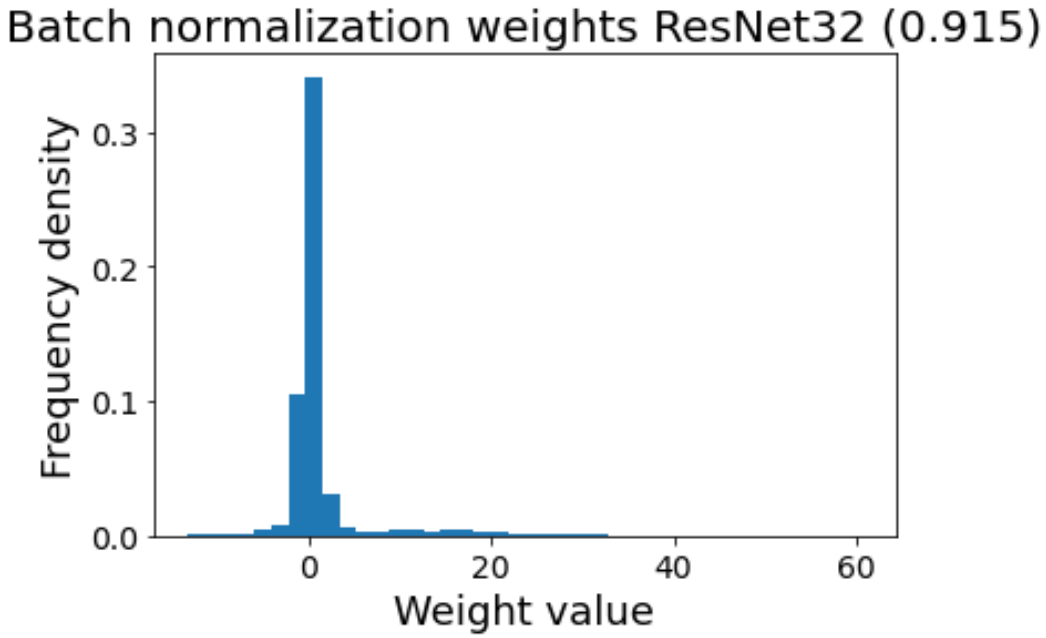

Figure 4.4. Histogram for quantitative distribution of batch normalization weights in trained ResNet32 with accuracy of 0.915.

5. CONCLUSION AND FUTURE WORKS

A recovery algorithm was presented for recovering the data corruption in the case of SEU. The ResNet32 and ResNet110 CNNs 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 ResNet32. The Apriori and FP-growth algorithms were used for finding the frequent patterns and subsequent rule mining. The tuned minimum support level and minimum confidence value thresholds were used later to define a probabilistic measure for identifying the corruption likelihood possibility.

The pre-trained parameters were 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, the number of significant bits was selected to be 9, 11, and 13 to investigate its effect on the recovery algorithm. The performance of recovery algorithm (accuracy in corruption prediction) 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 on a random bit. Moreover, in some bits, more than 99% of all the parameters were found to have one specific value (1 or 0). The performance of the algorithm was assessed in the case of corruption for those bits as well. For some bits, the recovery rate was noticed to be in the range of 98% to 100%, while for some other frequent bits and specially for ResNet110, the performance declined as low as 74%.

In this thesis, 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 can be of interest. While this work was focused on the recovery of bit corruption, rather than corrupted weight detection (with the assumption that the corrupted weight is already identified), future works can be conducted to develop algorithms for simultaneous detection of corrupted weights and corresponding corrupted bits.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, W. Xu, Cnn-rnn: A unified framework for multi-label image classification, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2285–2294.

[6] 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.

[7] R. Girshick, Fast r-cnn, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[8] X. Sun, P. Wu, S. C. Hoi, Face detection using deep learning: An improved faster rcnn approach, Neurocomputing 299 (2018) 42–50.

[9] 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.

[10] 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.

[11] H. Noh, S. Hong, B. Han, Learning deconvolution network for semantic segmentation, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1520–1528.

[12] D. Bolya, C. Zhou, F. Xiao, Y. J. Lee, Yolact: Real-time instance segmentation, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 9157–9166.

[13] E. Xie, P. Sun, X. Song, W. Wang, X. Liu, D. Liang, C. Shen, P. Luo, Polarmask: Single shot instance segmentation with polar representation, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 12193–12202.

[14] Y. Lee, J. Park, Centermask: Real-time anchor-free instance segmentation, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 13906–13915.

[15] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, Vol. 1, MIT press Cambridge, 2016.

[16] A. Khan, A. Sohail, U. Zahoora, A. S. Qureshi, A survey of the recent architectures of deep convolutional neural networks, Artificial Intelligence Review 53 (8) (2020) 5455–5516.

[17] J. Sanders, E. Kandrot, CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley Professional, 2010.

[18] 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.

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

[20] 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.

[21] 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.

[22] E. S. Marquez, J. S. Hare, M. Niranjan, Deep cascade learning, IEEE transactions on neural networks and learning systems 29 (11) (2018) 5475–5485.

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

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

[25] S. Marcel, Y. Rodriguez, Torchvision the machine-vision package of torch, in: Proceedings of the 18th ACM international conference on Multimedia, 2010, pp. 1485–1488.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, arXiv preprint arXiv:1912.01703.

[27] T. A. Henzinger, J. Sifakis, The embedded systems design challenge, in: International Symposium on Formal Methods, Springer, 2006, pp. 1–15.

[28] 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.

[29] V. Viswanathan, R. Hussein, Applications of image processing and real-time embedded systems in autonomous cars: a short review, International Journal of Image Processing (IJIP) 11 (2) (2017) 35.

[30] M. A. Hoque, C. Davidson, Design and implementation of an IoT-based smart home security system, International Journal of Networked and Distributed Computing 7 (2) (2019) 85–92.

[31] S. Turabzadeh, H. Meng, R. M. Swash, M. Pleva, J. Juhar, Facial expression emotion detection for real-time embedded systems, Technologies 6 (1) (2018) 17.

[32] 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).

[33] 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.

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

[35] 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.

[36] B. K. Dass, On a burst-error correcting code, Journal of information and Optimization Sciences 1 (3) (1980) 291–295.

[37] F. Wang, V. D. Agrawal, Single event upset: An embedded tutorial, in: 21st International Conference on VLSI Design (VLSID 2008), IEEE, 2008, pp. 429–434.

[38] A. Das, N. A. Touba, Low complexity burst error correcting codes to correct MBUs in SRAMs, in: Proceedings of the 2018 on Great Lakes Symposium on VLSI, 2018, pp. 219–224.

[39] C. Constantinescu, I. Parulkar, R. Harper, S. Michalak, Silent data corruption—myth or reality?, in: 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), IEEE, 2008, pp. 108–109.

[40] M. Ramzanpour, S. A. Ludwig, Association rule mining based algorithm for recovery of silent data corruption in convolutional neural network data storage, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2020, pp. 3057–3064.

[41] 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.

[42] A. Avizienis, Faulty-tolerant computing: An overview, Computer 4 (1) (1971) 5–8.

[43] C. Krishna, A. Singh, Reliability of checkpointed real-time systems using time redundancy, IEEE Transactions on Reliability 42 (3) (1993) 427–435.

[44] M. Nicolaidis, Design for soft error mitigation, IEEE Transactions on Device and Materials Reliability 5 (3) (2005) 405–418.

[45] S. Mitra, N. Seifert, M. Zhang, Q. Shi, K. S. Kim, Robust system design with built-in soft-error resilience, Computer 38 (2) (2005) 43–52.

[46] K. A. LaBel, M. M. Gates, A. K. Moran, P. W. Marshall, J. Barth, E. Stassinopoulos, C. M. Seidleck, C. J. Dale, Commercial microelectronics technologies for applications in the satellite radiation environment, in: 1996 IEEE Aerospace Applications Conference. Proceedings, Vol. 1, IEEE, 1996, pp. 375–390.

[47] A. B. Carlson, P. B. Crilly, Communication systems, 5e (2010).

[48] S. B. Wicker, V. K. Bhargava, Reed-Solomon codes and their applications, John Wiley & Sons, 1999.

[49] W. L. Pritchard, J. A. Sciulli, Satellite communication systems engineering, Prentice Hall, 1986.

[50] U. Legat, A. Biasizzo, F. Novak, Self-reparable system on FPGA for single event upset recovery, in: 6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), IEEE, 2011, pp. 1–6.

[51] A. Ejlali, B. M. Al-Hashimi, M. T. Schmitz, P. Rosinger, S. G. Miremadi, Combined time and information redundancy for seu-tolerance in energy-efficient real-time systems, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 14 (4) (2006) 323–335.

[52] V. P. Nelson, Fault-tolerant computing: Fundamental concepts, Computer 23 (7) (1990) 19–25.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

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

[60] D. Hunyadi, Performance comparison of Apriori and FP-Growth algorithms in generating association rules, in: Proceedings of the European computing conference, 2011, pp. 376–381.

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

[62] S. Raschka, Mlxtend (2016).

[63] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to data mining, Pearson Education India, 2016.

[64] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images.