# Recovery Algorithm to Correct Silent Data Corruption of Synaptic Storage in Convolutional Neural Networks

Arighna Roy and Simone A. Ludwig

*Department of Computer Science, North Dakota State University, Fargo, ND, USA,*
{*arighna.roy,simone.ludwig*}*@ndsu.edu*

**Abstract**

With the surge of computational power and efficient energy consumption management on embedded devices, embedded processing has grown exponentially during the last decade. In particular, computer vision has become prevalent in real-time embedded systems, which have always been a victim of transient fault due to its pervasive presence in harsh environments. Convolutional Neural Networks (CNN) are popular in the domain of embedded vision (computer vision in embedded systems) given the success they have shown. One problem encountered is that a pre-trained CNN on embedded devices is vastly affected by Silent Data Corruption (SDC). SDC refers to undetected data corruption that causes errors in data without any indication that the data is incorrect, and thus goes undetected. In this paper, we propose a software-based approach to recover the corrupted bits of a pre-trained CNN due to SDC. Our approach uses a rule-mining algorithm and we conduct experiments on the propagation of error through the topology of the CNN in order to detect the association of the bits for the weights of the pre-trained CNN. This approach increases the robustness of safety-critical embedded vision applications in volatile conditions. A proof of concept has been conducted for a combination of a CNN and a vision data set. We have successfully established the effectiveness of this approach for a very high level of SDC. The proposed approach can further be extended to other networks and data sets.

*Keywords:* Silent Data Corruption, CNN, AlexNet, Association Rule Mining

## 1. Introduction

The Artificial Intelligence (AI) problem space has become highly dependent on Machine Learning (ML) algorithms. AI is an interdisciplinary scientific field that is used by computer systems to build predictive models. One subfield trains a computer system to find patterns from historical data. One of the most popular branches of machine learning is Artificial Neural networks (ANN), which has become the state-of-the-art in many AI application areas. ANN is based upon the idea that a machine can learn patterns from data in a similar fashion

to how a human brain performs that task [1]. The concept of ANN was first introduced in 1943. The strength of ANN comes at the price of high computational complexity. This is the reason why ANN did not get much attention until the mid-80s.

Convolutional Neural Network (CNN), which is a specific type of ANN, is mostly used for computer-vision related problems. Computer vision is a branch of Artificial Intelligence that is concerned with the automation of tasks related to human visual systems. CNN is one of the most resource-intensive ANN due to its complex topology and size. However, with the rise of Graphics Processing Units (GPU) and distributed computing, CNN started to receive attention applied to computer-vision related problems. In 2012, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), AlexNet (a specific type of CNN), established CNN to be the state-of-the-art method for image data.

Table 1: Complexity of different CNN Architectures

| **CNN** | **Year** | **Parameters** |
|---|---|---|
| AlexNet [3] | 2012 | 60M |
| Clarify [7] | 2013 | 65M |
| OverFeat [5] | 2013 | 70M |
| VGG [8] | 2014 | 135M |

High-performance clusters with GPU acceleration can take care of the training phase of a neural network [2], which consumes the largest share of the computation. However, even pre-trained neural networks demand high memory due to the large number of connections in each layer (input, intermediate results, and output) in the network. The 8-layer AlexNet needs 240 MB to store 61 million weights [3] in a 32-bit floating-point format, and it is ever-growing. Table 1 shows how the parameter size for different CNN architectures has grown over time. For example, the number of weights in AlexNet that was introduced in 2012 was 60 million; whereas VGG (Visual Geometry Group), introduced in 2014, has 135 million weights.

Embedded systems adopted computer vision using CNN in no time since the semiconductor industry has evolved drastically over the years. Real-time embedded systems have many applications of advanced Decision Support System (DSS). A few of the good example applications of such DSS are smart environments, fitness monitoring, and military missions [4]. We will narrow down our discussion to computer vision applications in embedded systems (embedded vision). Thus, a good example of such a system is the biometric authentication system implemented in the iPhone [5]. The theoretical background of this problem is facial recognition, which is a well-known computer vision application.

However, due to the need to accommodate a massive number of weights, even the testing phase of such networks becomes a bottleneck when it comes to embedded devices because of the memory space restrictions [6]. An embedded

device is dedicated to a specific task on mechanical or electrical systems [7], and are often challenged by real-time computing constraints such as inefficient power management [8]. Specialized architectures such as conv-SRAM (CSRAM) [9] has been developed to accommodate CNN in microcontrollers, which is the heart of embedded systems.

Among many real-time constraints, embedded systems suffer from data corruption due to reduced size and power supply [6]. Often, embedded systems have to operate in harsh environments, which makes them more prone to errors. For example, rugged embedded systems are designed especially for harsh environments [6].

Soft error is the predominant challenge in the semiconductor industry [10]. With the rise in the use of real-time embedded systems, off-the-shelf embedded processors have received a lot of attention lately. These systems are cursed with having to work in harsh environments. This leads to external radiation of ionizing particles terrestrial neutron, which is the major source of soft errors [11]. That, in turn, aggravates the problem of inconsistent supply voltage [12], which has been a challenge for embedded systems ever since [13]. When the charge disturbance crosses the fault-tolerance threshold of the hardware, the data state of the memory cell flips and an undetected error occurs.

There are two types of Soft Errors, Single Event Upset (SEU), and Burst errors. The scope of our experiments is limited to SEU. Burst errors involve errors in multiple bits simultaneously. Error detection becomes much more complex for this type of error. Fortunately, burst errors occur rarely [14]. We can detect SEU with the help of checksum or parity bits, however, the exact position of the error remains unknown, which makes it unrecoverable. SEU very often cause Silent Data Corruption (SDC), which is very prevalent in embedded systems, and thus, is the prime focus of discussion in this paper.

During the testing phase, CNNs calculate the dot product of the input for each layer and the corresponding pre-trained weights to generate the output of that layer, which is then propagated to the next layer and used as input. SDC can lead to major deviation of the result from the expected, which might lead to detrimental consequences in systems since embedded vision is used in many critical real-time decision support systems such as self-driving cars [15].

There has been significant research on making embedded systems hardware resilient to data corruption. However, most of the robust hardware has a higher power consumption. For example, 8T and 6T are two varieties of SRAM (Static Random-Access Memory), and the 8T SRAM cell is a more reliable storage cell than 6T SRAM at the cost of a 40% higher power consumption. SRAM is used for the cache memory in embedded devices [16].

## 2. Related Work

Contemporary work on minimizing the impact of soft errors in embedded-vision applications can be broadly classified into two types; the first type, which comprises most of the work in the area, tries to address the problem by making

error-resilient hardware using efficient power management and storage. The second type is focused on building a software-centric approach to recover the corrupted bits to reduce the overall impact. Our strategy is based on the latter category.

The Embedded Vision Engine (EVE) [17] is a specialized architecture aimed at implementing automotive vision applications (computer vision applied to Advanced Driver Assistance Systems (ADAS)). EVE provides a fully programmable architecture that is 4-12 times faster and 4-5 times more energy-efficient.

Intelligent Protection Against Silent (IPAS) output corruption [18] is an instruction replication procedure to reduce the impact of Silent Output Corruption (SOC) errors by protecting only vulnerable instructions. Numerous hardware have evolved to mitigate the need for an embedded vision application. An SRAM-embedded convolution architecture [9] is designed to overcome the shortcoming of the architecture designed for generic embedded machine learning classifiers. It was tested on LeNet-5 CNN trained on the MNIST data set. It uses voltage averaging on a 266×62 Conv-SRAM (CSRAM) array.

Even with the most sophisticated hardware, embedded systems experience a substantial amount of silent errors [19]. A software-enabled approach to recover the corrupted bits could be beneficial for that. A generic [20] [21] first step of that approach is to analyze and identify the fault propagation through the application. The majority of the work on this type consists of duplication of instructions [22] to bring robustness against the silent errors, which is not feasible for CNN based applications due to the size. To give an example of an application-oriented approach, Augusto [23] proposed a hierarchical approach to tackle uncertainties of the operational environment of Unmanned Aerial Vehicles (UAV). The gap between the frame rate of the on-board camera and the observed frame rate on-ground is utilized to build a resilient "UAV swarm".

Algorithm-Based Fault Tolerance (ABFT) [24] is a software-based approach where algorithm-specific techniques are leveraged to detect and repair silent errors. The foundation of this approach is based on the fact that not all silent errors have a significant impact on the performance of the application. Being able to classify the sector where the silent errors cross the tolerance threshold (impact error bound) narrows down the problem space and reduces the numeric calculation for recovery. Another set of ABFTs focuses on more generic numeric operations such as matrix operations. Some of them focus on the checksum and roll-back recovery methods.

An exciting work that has been done in the area is a hybrid approach of a hardware and software-based technique [20]. The method leverages a software approach to identify the sensitive bits and then uses a hardware approach (selective latch hardening) to guard those bits selectively. However, with increased uncertainty in the environment of embedded applications, it became more and more challenging to improve the hardware to guarantee the performance of the embedded vision application. A pure software-based recovery approach can suffice to serve that purpose.

In this paper, we investigate the sensitivity of SDC through SEU on the memory bits storing the pre-trained weights of a CNN and the type of layers

(fully connected and convolutional). Once we identify the sensitive bits, we propose an algorithm to find the association between those bits and to recover the corrupted bits of the memory cells based on the generated rules of the association. We will show the experimental results for the different error levels and recovery rates. There are various techniques to detect the errors that occur in a RAM, and we will use the method called checksum, which is very commonly used for the same. Checksum is derived from a block of data in order to detect errors that occurred or have been introduced during its storage.

## 3. Approach and Methodology

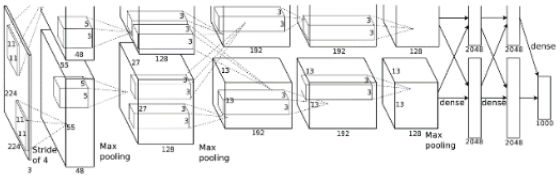
Association Rule Mining (ARM) [25] has been introduced in 1993. Since then, it has attracted considerable attention and has been applied in particular to market basket analysis for which customers' buying patterns are discovered from retail sales transactions. ARM is one of the key techniques to detect and extract useful information from data. ARM's aim is to identify strong rules discovered in the data by using some measures of "interestingness" [26]. In particular, to select interesting rules from the set of all possible rules, constraints on various measures of significance and interest are used; the best-known measures are minimum thresholds on support and confidence.

Unfortunately, the task of ARM is computationally expensive, in particular with large data set sizes as well as when large numbers of patterns exist. Thus, we have employed the FP-Growth Algorithm that was proposed by Han [27]. The FP-Growth algorithm is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth. The algorithm uses an extended prefix-tree structure for storing compressed and crucial information about the frequent patterns (named Frequent-Pattern tree (FP-tree)).

Figure 1 shows the overall process of building a recovery model, which starts with training a certain CNN on a certain image data set. Although the proposed model is generic to any CNN, it must be built on the weights of a pre-trained CNN to generate the association rules, which are the backbone of this model. Figure 2 provides a high level overview to replicate the process applied to other CNNs.

First, we have trained the Alexnet [28] on the CIFAR10 image data set [29], which yields a set of tensors. The tensors maintain the topology of the network; hence, those are flattened to combine all the weights. These weights can be denoted as ($W_1$, $W_2$, ..., $W_n$). Thereafter, each weight $W_i$ is represented as a series of thirty-two bits ($B_0$, $B_1$, ..., $B_3$). Hence, a matrix $W$ is created where each row signifies the binary representation of the float value of the corresponding weights. A cell from the matrix $W$ can be denoted as $W_{ij}$, which represents the $j^{th}$ bit of the $i^{th}$ weight.

First, we detect the bits that have the most impact on the accuracy of the CNN as MSB (Most Significant Bit) (msb0-m). Then we come up with deduction logic to prioritize the prediction. In our example, the $30^{th}$ and $29^{th}$ bits always contain certain values without any exception. In case of the prediction of the corrupted bits, we prioritize these bits to check the observed values. We use




Flatten the weights into a list and portray the bit representation

| | $B_{31}$ | $B_{30}$ | $B_{29}$ | ... | $B_1$ | $B_0$ |
|---|---|---|---|---|---|---|
| $W_1$ | 0 | 0 | 1 | ... | 0 | 1 |
| $W_2$ | 0 | 0 | 0 | ... | 0 | 0 |
| $W_3$ | 0 | 1 | 0 | ... | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... |
| $W_{n-1}$ | 1 | 0 | 1 | ... | 0 | 0 |
| $W_n$ | 0 | 0 | 1 | ... | 0 | 1 |

Filter the consecutive bits having the most impact on the performance of CNN

| | $B_7$ | $B_6$ | $B_5$ | $B_4$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
|---|---|---|---|---|---|---|---|---|
| $W_1$ | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| $W_2$ | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| $W_3$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| $W_{n-1}$ | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $W_n$ | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

Transform the bit values into transaction format

| | $i_7$ | $i_6$ | ... | $i_1$ | $i_0$ |
|---|---|---|---|---|---|
| $t_1$ | $i_{71}$ | $i_{61}$ | ... | $i_{11}$ | $i_{01}$ |
| $t_2$ | $i_{72}$ | $i_{62}$ | ... | $i_{12}$ | $i_{02}$ |
| ... | ... | ... | ... | ... | ... |
| $t_{n-1}$ | $i_{7n-1}$ | $i_{6n-1}$ | ... | $i_{1n-1}$ | $i_{0n-1}$ |
| $t_n$ | $i_{7n}$ | $i_{6n}$ | ... | $i_{1n}$ | $i_{0n}$ |

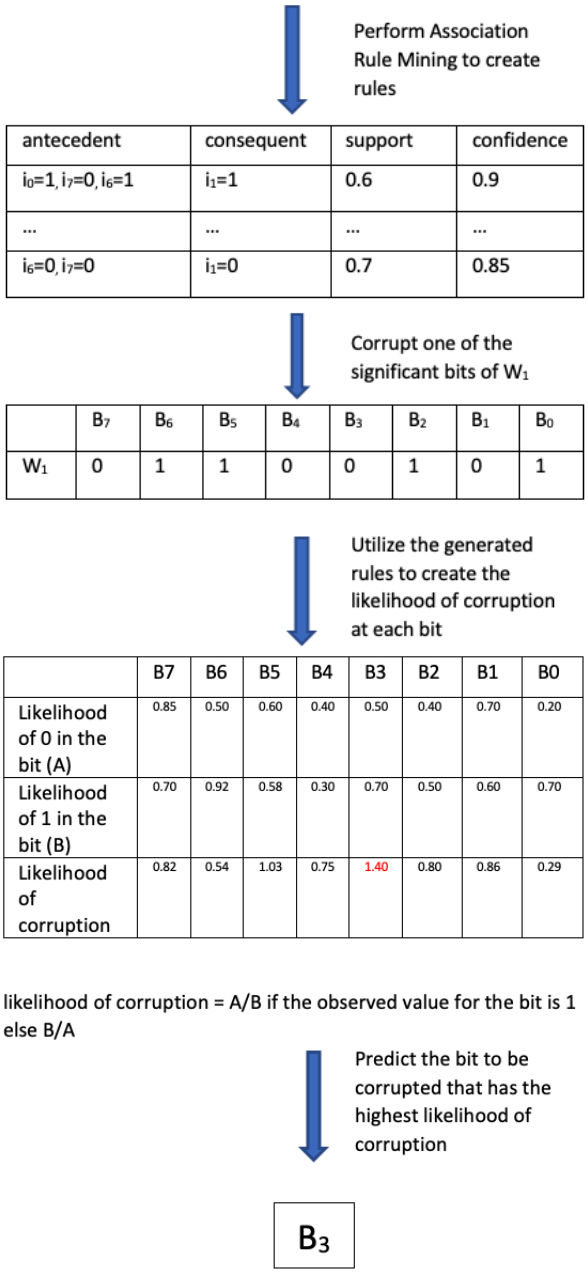



Figure 1: Flowchart of Methodology

a stochastic approach to build the rest of the recovery model, which is to find the association of the MSBs within these weights. We have used the FP-Growth algorithm in [30] to generate the association rules. The binary bit representation is considered as an individual item for this item set mining problem. Once we have the rules (for a certain min-confidence and min-frequency), we can build the recovery function. For a corrupted weight, each of these bits will be considered individually to calculate the likelihood of being corrupted. We have assumed that there is only one corrupted bit for each corrupted weight. For msbi, we find the rules that have msbi=0 in the consequent and the rest of the MSBs in the antecedent and calculate the sum of confidences of those rules. Similarly, we calculate the sum of confidences for msbi=1. If the bit value (0/1) with a higher likelihood does not match with the observed value for that weight, then it is considered to be the corrupted bit. We further check if there are multiple MSB that have this mismatch. For such scenarios, the ratio of the likelihood will be considered to predict the corrupted bit. The likelihood of a bit to be corrupted is calculated as the inverse ratio of the likelihood of the observed value of that bit.

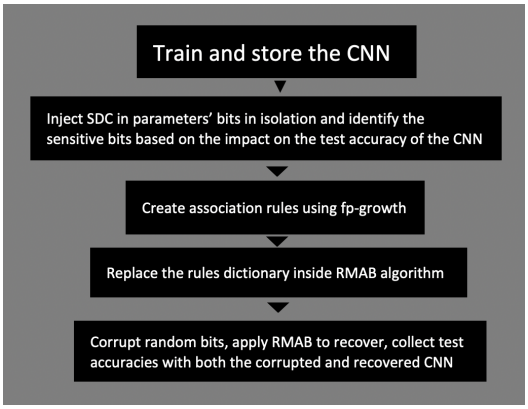



Figure 2: Process diagram to apply RMAB to other CNNs

Please note that if there are two bits that have the likelihood of corruption greater than zero, then that indicates that the observed value of that bit has a lower likelihood of occurrence. Hence, we chose the bit with the higher likelihood of corruption. In case of the tie (which is fairly rare), we select it randomly.

## 4. Experimental Setup and Results

We have performed various experiments on a pre-trained AlexNet to evaluate the effects of soft errors on the classification result. We first investigated how the soft errors propagate through the layers, and then narrowed down the potential area for recovery and applied the recovery model.

CNN stands out from the rest of the neural networks because various types of layers come together to build the network. Nevertheless, each layer has different shapes and sizes, even within the same type. That makes it inevitable to investigate it further and look deeper into the architecture of the network.

### *4.1. Structure of CNN (Alexnet)*

We have used the Alexnet for our experiments. Alexnet consists of five convolutional, five max-pooling, and one fully connected layer, which gives a total of thirteen layers, including the input and output layers. A pre-trained Alexnet has twelve sets of weight matrices; each set connects two consecutive layers. The following is the structure of an AlexNet: (64, 3, 11, 11), (64,), (192, 64, 5, 5), (192,) , (384, 192, 3, 3), (384,), (256, 384, 3, 3), (256,), (256, 256, 3, 3), (256,), (10, 256), (10,).

Each line of the above structure signifies a weight matrix. The last weight matrix, indicated by (10,) is to connect the last hidden layer to the output layer. The last hidden layer is fully connected, which is the decision layer. We have trained this model on the CIFAR10 data set, which has ten image classes. For obvious reasons, we have ten weights in the final set to connect ten nodes (each of them calculating the probability of the corresponding class for a specific input). (10, 256) indicates the weight matrix to connect the last Max-Pooling layer to the fully-connected layer.

The first weight matrix, indicated by (64, 3, 11, 11), is to connect the input to the first hidden layer, which is a convolutional layer. We have used 64 filters in this layer of size ($11 \times 11$). Each of these filters slides over the input matrix and collect the RGB values separately. The second index of this (first) weight matrix is always 3, as the images have depth three because of the three color channels.

The weight vectors followed by every 4-dimensional matrix are used to connect the convolutional layers to the following max-pooling layers. That explains why the first dimension of the 4-dimensional matrix always matches the length of the following vector. The rest of the 4-dimensional weight matrices are to connect the max-pooling layers to the next convolutional layers.

The following hyper-parameters were used for the experiments and training of the CNN:

- Number of Epochs = 164
- Learning Rate = 0.1
- Schedule = 81, 122

- Gamma = 0.1
- Optimizer = Stochastic Gradient Descent
- Momentum = 0.9
- Weight Decay = 5e-4

The accuracy of the trained CNN model is 77.22%.

### *4.2. Sensitivity of CNN to Soft Errors*

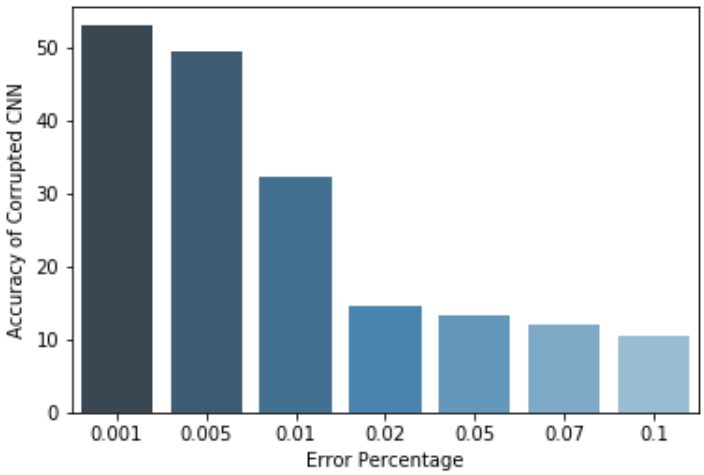



Figure 3: Accuracy of CNN after corruption vs. percentage of error

Figure 3 displays the accuracy of the Alexnet in the y-axis and the percentage of the soft error on the x-axis. The percentage of error is calculated based on the total number of weights in the network. Each weight is chosen randomly, and one of the 32 bits is randomly selected for each weight. We can see a sharp drop in the accuracy with the percentage of the soft error increasing until it reaches 0.05 and then it saturates. Please note that the accuracy of the trained model is 77.22%. In the later sections, we will observe that this behavior changes when we isolate either a layer or a bit index.

### *4.3. Sensitivity of Layers to Soft Errors*

Due to various layer types and sizes in the CNN, we have performed a layer level investigation. The layers vary significantly in size, which leads to a difference in impact on the accuracy. The larger a layer is, the more probable it is to be corrupted, which leads to a stronger influence on the accuracy.

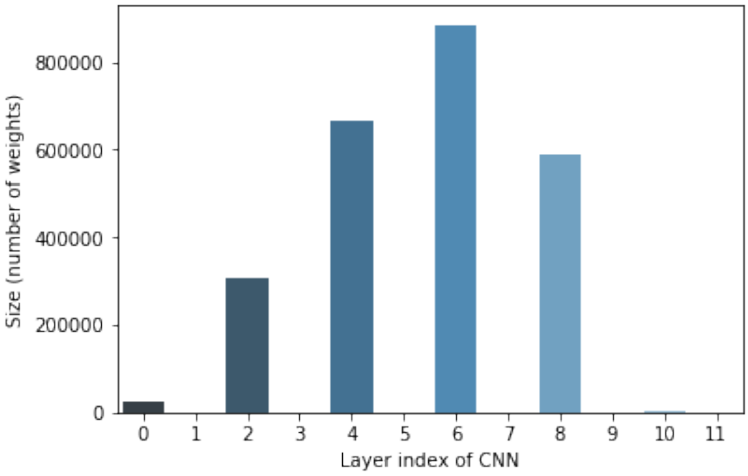



Figure 4: Number of weights within the layer vs. layer index

Figure 4 shows the size (number of weights it contains) of each layer. We can clearly see that only four of the layers dominate the volume of the weights, and all of those are convolutional layers. For further investigation of the layers, we will continue with the convolutional layers because of their sizes.

First, we examine the effect of the error on the classification accuracy when we fix a layer for the error injection. As part of this experiment, we have used the amount of error as the percentage of the layer size. This ensures to maintain a fixed likelihood of error occurrence for each layer while letting the error be generated for each layer in isolation.

Interestingly, the accuracy of the model does not depend on the size of the layer, as shown in Figure 5. Instead, we observe a clear pattern of increasing accuracy (or decreasing impact) towards the final layer of the neural network. This indicates that the impact of the level of features (on the classification accuracy) created at every convolutional layer decreases with the index. Thus, lower level features (for example, connections) formed at the earlier stage of the CNN are more critical than higher-level features (for example, shapes or objects) created in the later stages. For the experiments shown in Figure 5, we have used a random selection of bits. We have repeated the same experiment by isolating the bits to inject SDC at each layer and then averaged the accuracy over all the bits; the results are displayed in Figure 6. In the figure, we can clearly see that the accuracy of the CNN is not impacted at all when we isolate the bits. We will investigate individual bits next to identify the reason behind this behavior.

### *4.4. Sensitivity of Bits to SDC*

Our recovery model leverages the relationship of bit positions on the float value. We will first investigate the bit level dependency for the impact of SDC

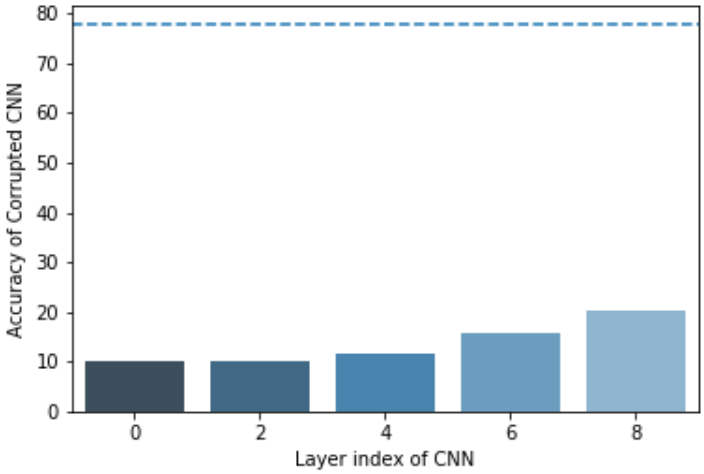



Figure 5: Accuracy of CNN after corruption (isolated to each layer) vs. layer index

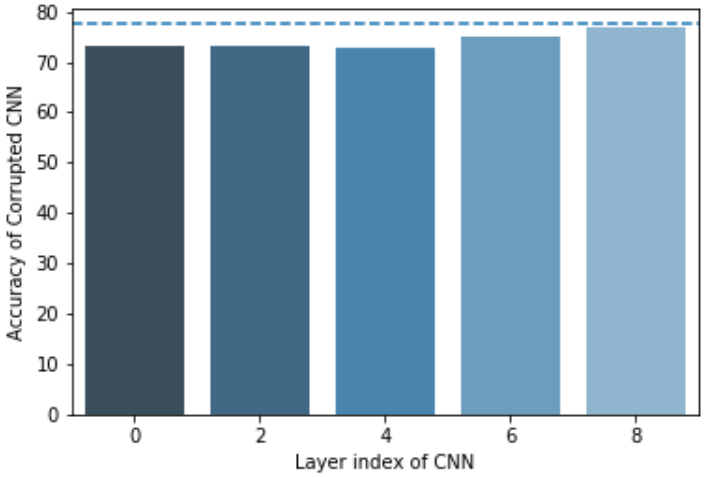



Figure 6: Accuracy of CNN after corruption (isolated to each layer and bit combination) vs. layer index

on the classification accuracy. This will help us to narrow down the problem in size. The recovery models dependent on selective hardware can also benefit from this. First, we will plot the frequency of 0/1 for each bit. From Figure 7 we observe that the $30^{th}$ bit (bit index starts from 0) always contains 0 and the $29^{th}$ bit always contains 1. Bits 3, 4, and 5 contain a significantly higher number of 1s compared to 0s. We will revisit this when looking at the layer-wise investigation.

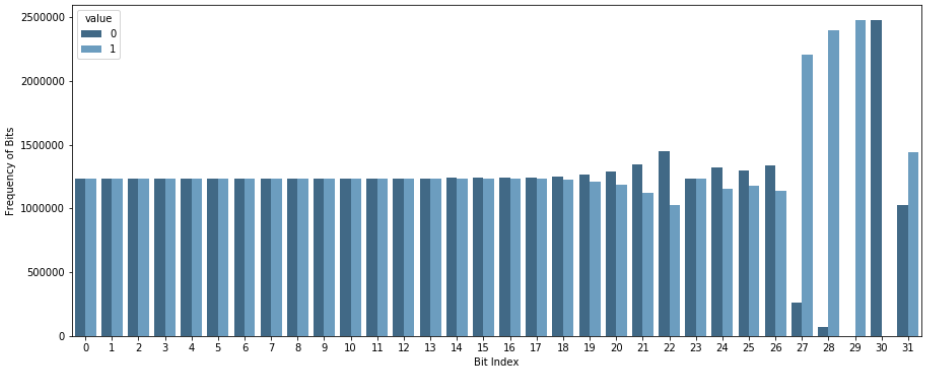



Figure 7: Binary value frequency vs. bit index

After that, we will investigate the impact of SDC on the classification accuracy for each bit. For this, we inject SDC on a specific bit of randomly chosen weights and calculate the classification accuracy. However, we will isolate each layer while injecting the SDC and then average the accuracy value later.

In Figure 8, we can clearly see that only a few bits are sensitive to the error. We will repeat our experiment for each bit with a randomly chosen layer (without isolating the layer). We did not find any pattern there itself. To investigate further, we chose random weights and random bits from the weight for each iteration for injecting the SDC at various levels of error volume.

Next, we will narrow down the area to be recovered for any kind of corruption. Only certain bits will be monitored using a checksum, which will be recovered. In Figure 8, we see that before bit 25 there is not much impact of SDC on the accuracy of CNN. We performed a controlled experiment on the accuracy of CNN with SDC injected on the last 7 bits ($25^{th} - 31^{st}$). In each iteration, the bit and the weight were chosen randomly.

Figures 9 and 10 show that the impact of SDC is limited to certain bits. Figure 9 showed the accuracy of the CNN when the SDC were restricted to the MSBs (25-31), and Figure 10 is for the LSBs (0-24). We can focus on protecting only the first 7 bits since the accuracy will not be affected by the SDC occurring on other bits. However, none of the experiments show any significant correlation with the percentage of error or the amount of SDC injected. The runtime cost

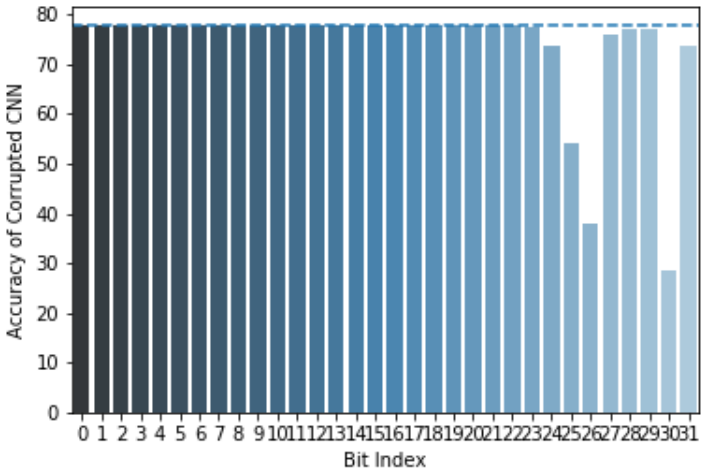



Figure 8: Accuracy of CNN after corruption (random layer) vs. bit index

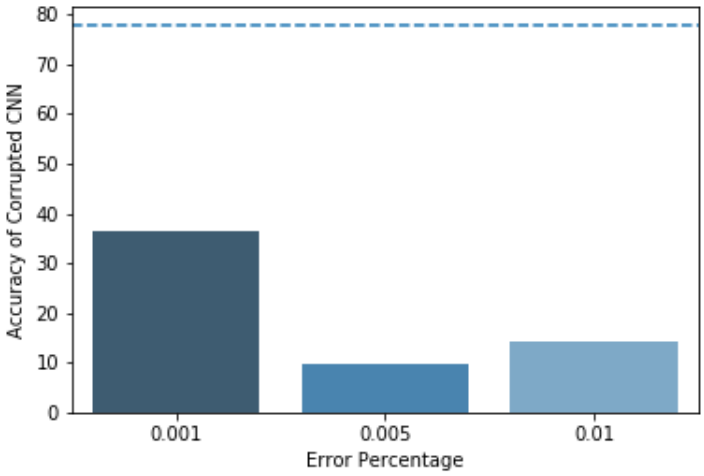



Figure 9: Accuracy of CNN after corruption (isolated to randomly chosen MSB) vs. percentage of error

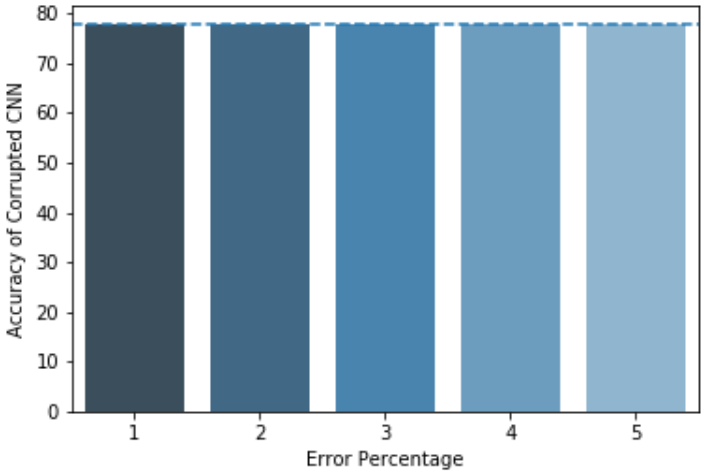



Figure 10: Accuracy of CNN after corruption (isolated to randomly chosen LSB) vs. percentage of error

of ARM increases exponentially with the number of bits considered. If we can narrow down the bits having a significant impact, the runtime cost of RMAB reduces drastically.

*4.5. Performance of Recovery Function on Significant Bits*

We maintained a checksum bit for Bit 0-7 to detect any occurrence of SDC. Once an error is detected on these bits, the $29^{th}$ and $30^{th}$ bit values are checked as these two bits always have certain values. If Bit 29 and Bit 30 do not have the expected values in the corrupted weight, the recovery algorithm is applied to predict the bit that is most likely to have been corrupted with the combination of the other bits.

First, we test the performance of the recovery model when the SDCs were injected on the MSBs being isolated. The results are shown in Figure 11. Bit 29 and Bit 30 are removed from the figure since those are recovered using deductive logic. However, those were considered for error injection. We see that the model does not perform so well when the SDC is injected on Bit 31. However, when we inject the error in randomly chosen bits (Bits 25 to 28) as shown in Figure 12, we achieve high performance after recovery. The performance of the corrupted model received a lot of impact even for such a low error rate. Although the accuracy of the corrupted model does not show a clear trend, the intention of the experiment was to confirm a high accuracy of the recovered model.

*4.6. Performance of Recovery Function on Convolutional Layers*

When we repeat the experiments of recovery for isolated layers, we see that the accuracy values are successfully reached compared to the original model

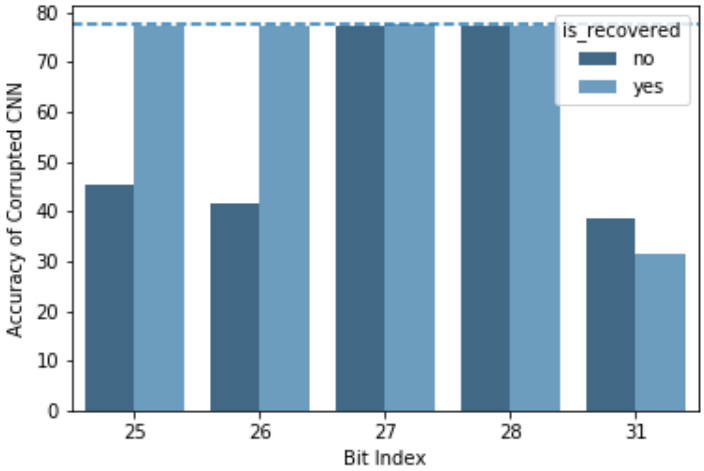



Figure 11: Accuracy of CNN after corruption (random layer) with or without recovery vs. bit index (only MSB)

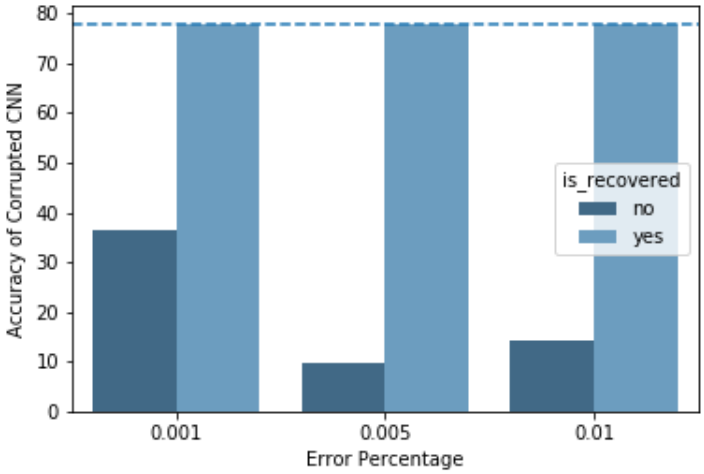



Figure 12: Accuracy of CNN after corruption (isolated to randomly chosen MSB) with or without recovery vs. percentage of error

irrespective of the layer (refer to Figure 13).

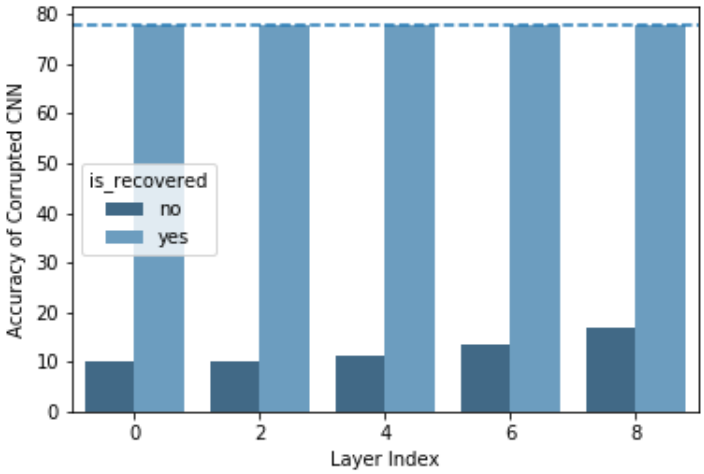



Figure 13: Accuracy of CNN after corruption (isolated to each layer) vs. layer index

### *4.7. Holistic Performance of Recovery Function*

The real evaluation of the recovery model can be tested with the random selection of bits and weights, where the accuracy of the model was impacted the most. In Figure 14, we can see that the performance slightly decreases (monotonously) with an increasing amount of SDCs. However, even with a 20% corruption rate (which is enormous in an embedded system environment), the recovery performance is very good.

The result of each experiment is averaged over thirty iterations to test the stability of the experimental results. The standard deviation of the accuracy through the iterations is plotted in Figure 15. The low standard deviation values of the accuracy for the experiments confirm the stability of the recovery model. Even though the recovery model did not perform well when the SDCs were injected on Bit 31 isolated, it performed well when the bits were randomly selected, which was the major weak area for the corrupted models in embedded systems.

## 5. Conclusion and Future Work

This paper addressed the problem of Silent Data Corruption (SDC) on a pre-trained CNN in embedded systems. The proposed approach employed an association rule mining algorithm to correct data corruption. We performed controlled experiments on the propagation of errors through the topology of the AlexNet trained on the CIFAR10 data set. Further investigations were

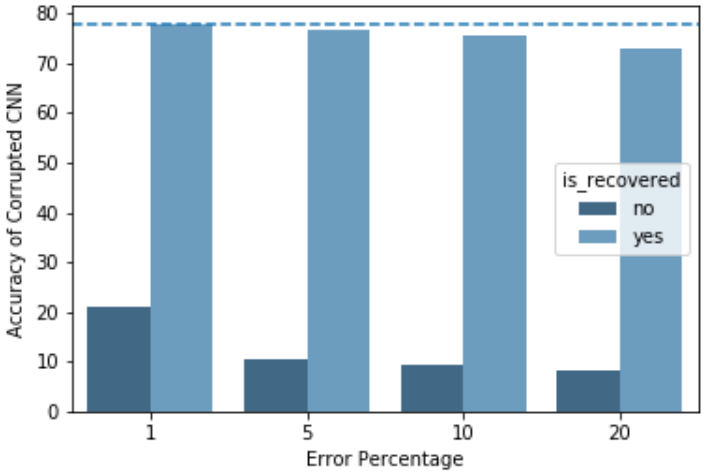



Figure 14: Accuracy of CNN after corruption with or without recovery vs. percentage of error

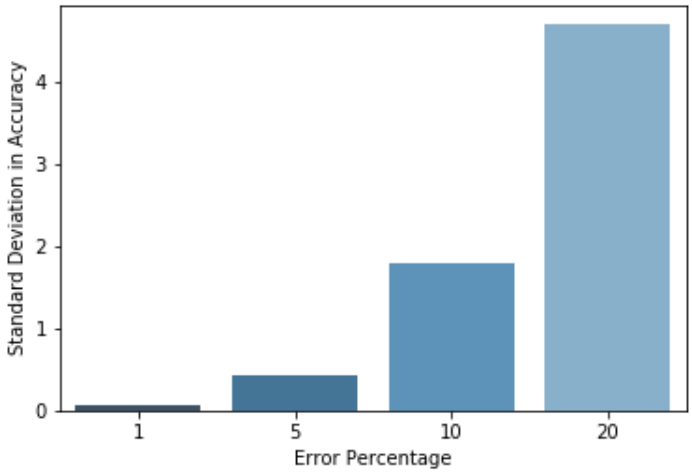



Figure 15: Standard deviation accuracy values of CNN for multiple experimental iterations vs. percentage of error

conducted to identify the sensitive segments of data based on its impact on the model performance. Thus, we converted the weights of the pre-trained network into a set of items (binary strings), and generated rules to detect the association among the bit values. These rules are then further used to build a recovery system. For each bit, the likelihood of containing a zero or one, depending on the state of the remaining bits, were calculated. Finally, we performed controlled experiments on the recovery algorithm to demonstrate the performance with various parameters.

The experiments establish the effectiveness of the recovery algorithm at various levels of data corruption. We have observed that the impact of SDC is much higher on the performance of the model when we target multiple bits for corruption on various weights (single bit for each weight, though). The proposed method successfully recovers the errors, and thus, the performance of the model does not get affected much even with high levels of corruption.

As for future work, we can extend this work to other CNN models such as VGG, FaceNet, and with larger image data sets such as CIFAR100. With each combination of CNN and data set, a new pre-trained model will be prepared for data corruption, then the sensitive section of the bit level storage must be identified, and the recovery model needs to be rebuilt based on the new rules of associations among the sensitive bits.

## Acknowledgment

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.

## References

[1] K. Kumar, G. S. M. Thakur, Advanced applications of neural networks and artificial intelligence: A review, IJ Information Technology and Computer Science 6 (2012) 57–68.

[2] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, A. Y. Ng, Large scale distributed deep networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1223–1231.
URL http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf

[3] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv preprint arXiv:1510.00149.

[4] T. Choudhury, G. Borriello, S. Consolvo, D. Haehnel, B. Harrison, B. Hemingway, J. Hightower, K. Koscher, A. LaMarca, J. A. Landay, et al., The mobile sensing platform: An embedded activity recognition system, IEEE Pervasive Computing 7 (2) (2008) 32–41.

[5] A. Rattani, R. Derakhshani, On fine-tuning convolutional neural networks for smartphone based ocular recognition, in: 2017 IEEE International Joint Conference on Biometrics (IJCB), IEEE, 2017, pp. 762–767.

[6] V. Narayanan, Y. Xie, Reliability concerns in embedded system designs, Computer 39 (1) (2006) 118–120.

[7] E. A. Lee, What's ahead for embedded software?, Computer 33 (9) (2000) 18–26.

[8] E. A. Lee, Cyber physical systems: Design challenges, in: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), IEEE, 2008, pp. 363–369.

[9] A. Biswas, A. P. Chandrakasan, Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications, in: 2018 IEEE International Solid-State Circuits Conference-(ISSCC), IEEE, 2018, pp. 488–490.

[10] R. C. Baumann, Soft errors in commercial integrated circuits, International Journal of High Speed Electronics and Systems 14 (02) (2004) 299–309.

[11] R. C. Baumann, Radiation-induced soft errors in advanced semiconductor technologies, IEEE Transactions on Device and materials reliability 5 (3) (2005) 305–316.

[12] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, Soft error and energy consumption interactions: A data cache perspective, in: Proceedings of the 2004 international symposium on Low power electronics and design, ACM, 2004, pp. 132–137.

[13] F. Mueller, Challenges for cyber-physical systems: Security, timing analysis and soft error protection, in: High-Confidence Software Platforms for Cyber-Physical Systems (HCSP-CPS) Workshop, Alexandria, Virginia, 2006, p. 4.

[14] J. Maiz, S. Hareland, K. Zhang, P. Armstrong, Characterization of multi-bit soft error events in advanced srams, in: IEEE International Electron Devices Meeting 2003, IEEE, 2003, pp. 21–4.

[15] J. Kim, H. Shin, Algorithm & SoC Design for Automotive Vision Systems, Springer, 2014.

[16] G. Srinivasan, P. Wijesinghe, S. S. Sarwar, A. Jaiswal, K. Roy, Significance driven hybrid 8t-6t sram for energy-efficient synaptic storage in artificial neural networks, in: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2016, pp. 151–156.

[17] D. K. Mandal, J. Sankaran, A. Gupta, K. Castille, S. Gondkar, S. Kamath, P. Sundar, A. Phipps, An embedded vision engine (eve) for automotive vision processing, in: 2014 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2014, pp. 49–52.

[18] I. Laguna, M. Schulz, D. F. Richards, J. Calhoun, L. Olson, Ipas: Intelligent protection against silent output corruption in scientific applications, in: 2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), IEEE, 2016, pp. 227–238.

[19] G. Shi, J. Enos, M. Showerman, V. Kindratenko, On testing gpu memory for hard and soft errors, in: Proc. Symposium on Application Accelerators in High-Performance Computing, Vol. 107, 2009.

[20] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, S. W. Keckler, Understanding error propagation in deep learning neural network (dnn) accelerators and applications, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, p. 8.

[21] R. A. Ashraf, R. Gioiosa, G. Kestor, R. F. DeMara, C.-Y. Cher, P. Bose, Understanding the propagation of transient errors in hpc applications, in: SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2015, pp. 1–12.

[22] N. Oh, P. Shirvani, E. Mccluskey, Error detection by duplicated instructions, IEEE Transactions on Reliability (2).

[23] A. Vega, C.-C. Lin, K. Swaminathan, A. Buyuktosunoglu, S. Pankanti, P. Bose, Resilient, uav-embedded real-time computing, in: 2015 33rd IEEE International Conference on Computer Design (ICCD), IEEE, 2015, pp. 736–739.

[24] K.-H. Huang, et al., Algorithm-based fault tolerance for matrix operations, IEEE transactions on computers 100 (6) (1984) 518–528.

[25] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in: Acm sigmod record, Vol. 22, ACM, 1993, pp. 207–216.

[26] G. Piatetsky-Shapiro, Discovery, analysis, and presentation of strong rules, Knowledge discovery in databases (1991) 229–238.

[27] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: ACM sigmod record, Vol. 29, ACM, 2000, pp. 1–12.

[28] H. M. Bui, M. Lech, E. Cheng, K. Neville, I. S. Burnett, Object recognition using deep convolutional features transformed by a recursive network structure, IEEE Access 4 (2016) 10059–10066.

[29] B. Recht, R. Roelofs, L. Schmidt, V. Shankar, Do cifar-10 classifiers generalize to cifar-10?, arXiv preprint arXiv:1806.00451.

[30] C. Borgelt, An implementation of the fp-growth algorithm, in: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations, ACM, 2005, pp. 1–5.