IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 28, 2021, accepted April 22, 2021, date of publication April 29, 2021, date of current version May 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3076716

A Selective Mitigation Technique of Soft Errors for DNN Models Used in Healthcare Applications: DenseNet201 Case Study

KHALID ADAM[1], **IZZELDIN IBRAHIM MOHAMED**[1], **AND YOUNIS IBRAHIM**[2]
[1]College of Engineering, University Malaysia Pahang, Kuantan 26300, Malaysia
[2]College of IoT Engineering, Hohai University, Changzhou 213022, China

Corresponding author: Khalid Adam (khalidwsn15@gmail.com)

This work was supported by Universiti Malaysia Pahang, through the UMP internal under Grant RDU1903149.

ABSTRACT Deep neural networks (DNNs) have been successfully deployed in widespread domains, including healthcare applications. DenseNet201 is a new DNN architecture used in healthcare systems (i.e., presence detection of the surgical tool). Specialized accelerators such as GPUs have been used to speed up the execution of DNNs. Nevertheless, GPUs are prone to transient effects and other reliability threats, which can impact DNN models' reliability. Safety-critical systems, such as healthcare applications, must be highly reliable because minor errors might lead to severe injury or death. In this paper, we propose a selective mitigation technique that relies on in-depth analysis. First, we inject the DenseNet201 model implemented on a GPU via NVIDIA's SASSIFI fault injector. Second, we perform a comprehensive analysis from the perspective of kernel and layer to identify the most vulnerable portions of the injected model. Finally, we validate our technique by applying it to the top-vulnerable kernels to selectively protect the only sensitive portions of the model to avoid unnecessary overheads. Our experiments demonstrate that our mitigation technique achieves a significant reduction in the percentage of errors that cause malfunction (errors that lead to misclassification) from **6.463%** to **0.21%**. Moreover, the performance overhead (the execution time) of our technique is compared with the well-known protection techniques: Algorithm-Based Fault Tolerance (ABFT), Double Modular Redundancy (DMR), and Triple Modular Redundancy (TMR). The proposed solution shows only **0.3035%** overhead compared to these techniques while correcting up **84.8%** of the SDC errors in DenseNet201, remarkably improving the healthcare domain's model reliability.

INDEX TERMS Convolutional neural networks, DenseNet201, healthcare, GPUs, soft error, reliability.

I. INTRODUCTION

Convolutional Neural Networks (CNNs) became prominent following the study conducted by LeCuN, where it was used to process grid-like data of images and time series [1]. Since then, CNNs have been observed to be among the most preferred approaches that may be used to understand images' content. CNNs are a special type of Deep Neural Networks (DNNs), which have shown state-of-the-art results on many competitive benchmarks. In fact, reports have revealed that DNNs present highly desirable outcomes when used for tasks such as image recognition [2], segmentation [3], detection [4], as well as retrieval. Based on this, the interests in DNNs have extended beyond academia. Specifically, DNNs are now extensively applied in healthcare applications such as automatic recognition of surgical [5], pathological brain detection [6], and robotic surgery (e.g., Da Vinci) [7], among many others. Due to the massive parallelism structure, DNNs require high computational capabilities such as Graphics Processing Units (GPUs) [8].

Consequently, graphics processing units (GPUs) are extensively used nowadays in DNN models to overcome the inherent computational challenges of healthcare applications [9]–[11]. Interestingly, computation through GPU has offered impeccable advantages over other accelerators [12]. Notwithstanding, there are specific GPU units that, if exposed to soft errors, can disrupt the reliability of the GPU operations; these units include memory elements such as register file and logic resources such as Arithmetic Logic Units (ALUs) [13]. Hence, when using GPUs in healthcare,

The associate editor coordinating the review of this manuscript and approving it for publication was Shen Yin.

it is crucial to ensure that potential data corruption is avoided and failure rates must be reduced to the minimum [13]. For instance, Food and Drug Administration (FDA) reported that 1078 of the adverse events (10.1%) were unintended errors (soft errors) that happened, including 52 injuries and two deaths [14]. To the best of our knowledge, none of the previous studies considered the reliability of the healthcare system that was affected by soft errors in their analysis, and none of them assessed the risk of adverse events across different healthcare applications and their impact. The worse behavior of these errors is that they occur without stopping the program, but the output will be different once the program is finished. Therefore, it becomes vital to evaluate the DNN models' resilience that runs on the GPU [15].

Although several studies [16], [8], [17], [18], [17] have evaluated and analyzed the reliability of DNN models, and many techniques have been proposed to mitigation the soft errors in GPUs based on software solutions. For instance, Triple Modular Redundancy (TMR), Double Modular Redundancy (DMR), and Algorithm-Based Fault Tolerance (ABFT). Firstly, these techniques produce high runtime overheads. Secondly, the behavior and workflows of the vast majority of DNNs architectures are different. Thirdly, these DNNs are utilized on a broad range of accelerators, all of which have different components and peculiar execution flows. Thus, it is not easy to directly apply a specific DNN's case to solve other architectures [15]. In this paper, the DNNs model's reliability (DenseNet201) run on GPU was analyzed by extensive fault-injection campaigns that were carried out on the GPU to examine its reliability. The main contributions are as follows:

- An extensive analysis of DenseNet201 characteristics under soft errors (SASSIFI fault-injection).
- Thorough analysis of SDCs (i.e., Malfunction and Light-Malfunction) errors in the kernels of DenseNet201 to determine the vulnerable part of the model.
- Development of selective-based mitigation solution for the DNNs model and validating it with DenseNet201.
- Comparing the proposed technique's performance overhead with three protection techniques: ABFT, DMR, and TMR is carried.

The remainder of this paper is organized as follows. Section II discusses motivation and related work to show the key concepts of DNN reliability and how our work is different. Section III presents DenseNet201, DNNs accelerators used in the healthcare system, and soft errors in GPUs. Section IV describes the dataset and training results. Section V provides the experimental setup. Section VI, discusses results and analyzes the resilience of DenseNet201. Section VII introduces the proposed solution and its validation. Section VIII performs overhead comparisons of our selective mitigation technique with other techniques. Finally, Section XI presents conclusions and outlines directions for future research.

II. MOTIVATION AND RELATED WORK

The authors' motivation to analyze the DNN models' reliability and the available techniques to mitigate the soft error problems is based on various factors. Firstly, even though several studies have addressed the reliability under soft errors, none of the studies focused on analyzing the impact of resilience error in DenseNet201 on the GPU. Secondly, DNN models' reliability depends on many factors such as the network topology, layers positions, and layers number [8]. For example, we analyze DenseNet201, which has 201 convolutional layers, 98 route layers, one avg pooling, 4 max-pooling, and Softmax. Thirdly, as healthcare applications (i.e., presence detection of the surgical tool) are safety-critical systems, they should be highly reliable because small errors might lead to serious injury or death [19]. Besides, it is well known that the reliability of the system/device refers to the resilience of such devices against vulnerabilities to faults in the electronic components. As such, failures are generally rare and are easily ignored for typical applications. However, for safety-critical applications such as presence detection of surgical tools in the operations, it is vital to consider the role of the underlying soft errors in system reliability [7], [20].

Therefore, in this contribution, previous studies on the reliability of DNN models were reviewed. Specifically, we addressed the DNNs reliability via DNN accelerators under soft errors. Very comprehensive work is presented by Beibei *et al.* [21]. They proposed a novel modulated anchoring network to detection surgery tools, and the network evaluated on the m2cai16 dataset. However, the authors verified the reliability by asked experts to manually assess the performance.

Santos *et al.* [16] analyzed three CNN architectures: YOLO, Fast R-CNN, and ResNet. However, YOLO and Fast R-CNN are object detection models, while ResNet is an image classification model. Although detection and classification are wildly different in their structures and workflows, the authors only analyzed the YOLO structure. Thus, they studied the ResNet (classification) model only by measuring the Precision and Recall of the corrupted output, without getting insights on how errors propagate through ResNet's layers or kernels. In our work, we studied the classification model (DenesNet201) in detail. One of the main differences is that there are Normalize layers in the DeseNet201 model, which are among the most vulnerable layers in our analysis, while these layers do not even exist in YOLO and Fast R-CNN. Moreover, in our study, we make a comprehensive analysis of the DensNet201 reliability, including Layers vulnerability analysis and Kernels vulnerability.

Santos *et al.* [22] studied two object detection systems, Histogram of Oriented Gradients (HOG) and You Only Look Once (YOLO). The concept of KVF is employed in HOG, while LVF for YOLO. The difference is that HOG has no layer in its structure because it is not a CNN-based algorithm, while YOLO does. However, the KVF concept has not been considered for YOLO. However, the authors stated, "*We proposed a smart layer duplication that detects more than 90%*

of errors, with an overhead lower than 60%." Consequently, their hardening strategy for CNN-based models is to harden the entire layer. In our work, on the other hand, we analyze DenseNet201's layers to identify the most vulnerable ones. Then within each layer, we analyze its kernels to determine the most vulnerable kernels. Thus, we apply the TMR approach to the kernels of the CNN-based models.

Islam *et al.* [23] put forward an approach to separate surgical instruments from high-resolution videos generated through a commercial robotic system (Da Vinci robot), based on a cascaded lightweight CNN. It is evident from the study reports that the proposed technique is suitable for use in tissue segmentation because the approach is significantly enhanced for the segmentation of robotic instruments. Nevertheless, the reliability of the approach was not considered. Based on the study by Li *et al.* [8], most of the important DNNs related concepts have been comprehensively analyzed, and it provides an in-depth understanding of the generation of errors in contemporary DNN systems. Information gathered from the study revealed that several factors could influence the reliability of DNNs. Notable among these factors are network topology, data types, positions, data reuses, layers numbers, and values. Four CNN architectures, including NiN, AlexNet, ConVet, and CaffeNet, were used in the study to discuss the issues of reliability in CNN models. However, an ASIC design, Eyeriss, a specific DNN accelerator, was used [24], But in real terms, the fault response/sensitivity of DNN accelerators differs. Therefore, there are two notable differences between their work and ours. Firstly, ASIC was used as the DNN accelerator in their study, whereas GPU is used in this current study. Secondly, a contemporary CNN architecture, which is much deeper (i.e., DenseNet201), is used in this present study.

In a more recent and closely related work by Y. Ibrahim *et al.* [15], the reliability of CNN (ResNet) on the GPUs was investigated. This study focuses on model depth. Thus, the authors mainly intended to determine the impact of using a component in a radioactive environment for a longer time for three ResNet models. Nevertheless, the authors do not study the critical SDC errors in each kernel. However, in our study, we perform the following studies (i) a modern and much deeper CNN architecture (DenseNet201) is analyzed, and (ii) the error propagation in DenseNet201 is studied in order to determine the critical error (i.e., Malfunction) within each kernel of the model. Another essential point is that injecting faults into the ResNet model provokes its Residual layers, significantly impacting their resilience more than the other layers, compared to route layers in DenseNet201 do not generate errors. Table 1 presents a summary of related studies concerning their focus and findings compared to this research paper.

As presented in Table 1, it is evident that most of the related previous studies mainly concentrated on the accuracy of DNN models in healthcare applications. In contrast, the reliability of DNN-based on healthcare systems is rarely investigated. Two studies out of works reviewed in Table 1 have presented close work to ours. Nevertheless, we still can tell what makes our work unique in a few sentences. First, the authors in [16] studied the only classification model (ResNet) in their paper only by measuring the Precision and Recall of the corrupted output, without getting insights into how errors propagate among ResNet's layers or kernels. This can be seen in the reason why Santos *et al.* considered "error propagation analysis" in their study, while the ResNet model was excluded. The difference is that the authors adopted algorithm-based fault-tolerance (ABFT), which is a hardening strategy for Matrix Multiplication (MxM). This means that ABFT only protects the MxM of the convolution layer while leaving out other layers, such as Normalize layer, which is one of the most vulnerable layers in modern CNN models.

Secondly, the authors in [15] focused on analyzing and evaluating error resilience in the three ResNet models. Although this study is also CNN-based, the ResNet has a very different architecture than DenseNet due to the Residual module. Moreover, this study focuses more on the impact of the model's depth, analyzing ResNet-50, ResNet-101, and ResNet-152. Therefore, this necessitates further studies, especially considering that the programming models used with GPU are very different, which can significantly influence the propagation of error in GPU-based DNN models. In other words, one cannot generalize a specific DNN's reliability case to other architectures. Besides, the behavior of soft errors in DNN models under GPU is highly application-specific [25]. However, this has not been considered by the previous studies. Herein, an empirical study was performed to figure out and characterize the propagation of error in the DNN model (DenseNet201). Specifically, a fault-injection was performed to the GPU applications to track the propagation of error in DenseNet201 easily. Then the reliability of the model under soft error propagation in GPU was evaluated. However, to the best of the authors' knowledge, this present study is the first to analyze the reliability of the classification model (DenseNet201) implemented on a GPU and investigate the emergence of misclassifications as a result of soft errors.

III. BACKGROUND

This section provides a brief background on the reliability of CNNs in healthcare applications, the DenseNet201 model, GPUs' architecture, and soft errors in GPUs and their impact on the application execution.

A. CONVOLUTIONAL NEURAL NETWORKS (CNNs)

The CNN technique is the most commonly used approach for Deep Learning (DL). Recently, they have been more commonly used in healthcare applications such as scanning, diagnosing, generating reports, and treating various diseases [26]–[30]. By using CNNs, these complex healthcare applications would be incorporated into the network, thereby acting as trainable feature extractors with some extent of scale, shift, as well as deformation invariance [19]. One of the major layers in this technique is the convolutional layer. The other two layers include the subsampling layer, which is optional,

TABLE 1. Summary of related studies.

Ref	Study focus	Similar to our work	Not consider
[21]	This proposed a novel modulated anchoring network to detect surgical tools.	CNN models run on top of GPU.	The authors only consider reliability by asking experts to assess the performance manually. Whereas we test the reliability of the model under fault injection (soft errors) to evaluate the reliability
[16]	This study analyzed the reliability of the three CNN architectures, namely YOLO, Fast R-CNN, and ResNet.	CNN models run on top of GPU.	This study makes a comprehensive analysis of the three CNN architectures' reliability, including Layers vulnerability analysis and Kernels vulnerability. We adopted TMR, and proposed a selective hardening solution based on the TMR to harden only the most vulnerable kernels by triplicating some instead of the whole model.
[22]	The authors in this paper studied two object detection systems, Histogram of Oriented Gradients (HOG) and You Only Look Once (YOLO).	The concepts of KVF and LVF were formalized in the paper.	The KVF is employed in HOG, while LVF for YOLO. However, HOG has no layer in its structure because it is not a CNN-based algorithm. We analyze DenseNet201's layers to identify the most vulnerable ones. Then within each layer, we analyze its kernels to determine the most vulnerable kernels. Thus, we apply the TMR approach to the kernels of the CNN-based models.
[23]	This study focused on surgical instruments from high-resolution videos obtained from a commercial robotic system (Da Vinci Xi robot).	CNN models run on top of GPU.	This study does not consider reliability. We used CNN model DenseNet201.
[8]	This study focused on Self-driving cars and used Eyeriss to accelerator explain reliability.	CNN models (e.g., AlexNet). Fault injection to evaluate reliability.	We considered faults that occur in other computing resources, such as logic units. We used GPUs where the logic units more sensitive to radiation-induced soft errors. We used CNN model DenseNet201.
[15]	This study analyzed and evaluated the error resilience of three ResNet models.	CNN models run on top of GPU.	We use a CNN model called DenseNet201. The critical errors in each kernel are determined and protected,

and the fully connected layers. The arrangement of these layers is in a feed-forward fashion [31].

The most notable architectures of the CNN technique are ResNet [32], VGGNet [33], GoogLeNet [34], AlexNet [35], and DenseNet201 [36]. Considering the safety-critical nature of the healthcare sector, applications used in healthcare must have high reliability. This is mainly because even small errors might lead to serious injury or death, as reported by the Food and Drug Administration (FDA) department [14].

B. DenseNet201

Dense Convolutional Network (DenseNet201) is a new deep CNNs architecture proposed by Huang *et al.* [36]. DenseNet201 present state-of-art CNNs is notable for its astonishing performance in benchmark tasks such as CIFAR-100 and ImageNet, which require competitive object recognition. Recent studies have shown that DenseNet is useful in healthcare applications such as diagnosis [37], medical image [39], anatomical Brain segmentation [39], and surgical [39] as it is considerably more accurate with fewer parameters. This model's performance hinged mainly on its ability to offer better parameters and training efficiency through the reuse of features. This is achieved by concerting preceding feature-maps that have been generated by previous layers and incorporating them into the successive layer. By so doing, it becomes possible for the feature-maps generated by previous layers to be easily accessed by the network deep layers so that the features can be reused, as illustrated in Figure 1. Hence, for operation with several layers equal to Xl, all feature-maps in preceding layers (X0,..., Xl− 1) can project the feature-maps in the lth layer (Xl) as follows:

$$Xl = Hl([X0, \ldots, Xl - 1]) \tag{1}$$

where $[X0, \ldots, Xl - 1]$ respectively represents the feature-maps in the $0th$, …, $l-\ 1th$ layer. The function, $Hl(\cdot)$, represents a composite function of three operations, which comprises the convolution (Conv), batch normalization (BN), and a rectified linear unit (ReLU). In conventional deep CNNs, the convolutional layers usually precede down-sampling layers, which reduce the height and width of

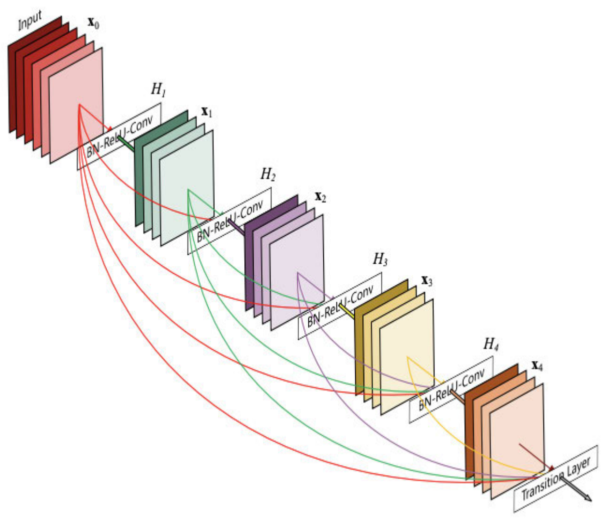


FIGURE 1. DenseNet201 architecture [37].

feature-maps by 50%. As a result of this, it would be challenging to concatenate feature-maps before downsampling layers and, subsequently, size differences. In order to overcome this challenge, craftily constructed dense blocks were made, with subsequent down-sampling layers, while dense block layers are densely linked. Based on this, there would be no change in the sizes of dense block feature-maps; howbeit they would have been reduced by half after down-sampling.

Therefore, taking the case of a dense block that comprises L layers, the overall number of directly linked layers is $L(L+1)/2$. On the contrary, in a convolutional network with L convolution layers, the number of connections is only L. However, there is a concurrent increase in the number of concatenated feature-maps of the successive layer as layers become deep. If constraints are not placed on the continuing growth in the number of feature-maps, there could be a disaster emanating from enormous computation expenditure. Therefore, the number of freshly generated feature-maps in each layer is controlled by designing the growth rate k. Due to this, in operation with an l number of layers, the *lth* layer in the dense block will have a total of $k\ x\ (l-1)+\ k0$ feature-maps where *k0* represents the number of input channels in the dense block.

DenseNet201 is made from four dense blocks consisting of 305 layers; each of these layers has one or more than one kernel. The process of generating new feature-maps and concatenating in blocks of DenseNet20 involves applying a convolutional (1×1 Conv, 3×3 Conv), where the filters are used 128 and 32 respectively with fixed input and output in each block started from 64×64, in the first block until reached 8×8 in the last block. Then, the Route layer only takes the preceding layers' results without processing to the next layer, followed by ReLU activation. Then, transition layers between two adjacent blocks, which do convolutional and Max-pooling (1×1 Conv., 2×2 Max), reduce feature maps' sizes. There will be a continuous repetition of this structure until just prior to the classifier, with a growth rate of feature-maps in each block1, block2, block3, and block4 are [6], [12], [47], [31] respectively. Then, the classification layer has a fully-connected (FC) layer that performs seven classes, followed by global average pooling, and finally, a softmax classifier is attached, as illustrated in Figure 2.

Furthermore, DenseNet201 consists of several types of layers that facilitate computation. Once the DenseNet algorithm is executed on the GPU, each layer will be composed of several kernels. There is vast diversity among these kernels, ranging from the required memory to the volume of instructions consisted of the kernels to the time required for their execution. DenseNet201 kernels pairs and their respective CNN architecture layers based on the Darknet framework are collected. To give an in-depth analysis of how different kernels have different vulnerability levels to soft errors and how they contribute to the final output of the model (i.e., classification of the objects), we need to find out all the static kernels needed for this task. Also, the kernels used for inferencing are considered. This means we do not include kernels used for training.

Notably, in Table 2, ten kernels each have a specific task during the model's execution computations. These ten kernels as follows: ***Im2col_gpu*** kernel represents the first step in transforming a convolutional operation to a matrix-multiplication operation, which is used in the original Caffe's convolution to perform matrix-type multiplications. This is achieved through the layout of all the patches and filters into matrices. Also, the ***Forward_avgpool*** kernel performs down-sampling in their corresponding layers. The ***Forward_maxpool*** kernel is usually placed after the Convolutional layer. The utility of the pooling layer is to reduce the spatial dimension of the input volume for the next layers. ***Add_bias*** kernel is used in convolutional or in fully-connected operations (layers) to add biases to the necessary parameters after the matrix multiplication in the learning process. ***Scale_bias*** kernel for dividing the input by its standard deviation to have a variance of approximately. ***Normalize*** kernel performs the normalization task for the GPU input. ***Copy*** kernel is for Normalize layers to feed the GPU buffer with the input, filters, and biases before the computations task. ***Activation_array,*** this kernel applies nonlinearity to the feature maps to reduce the input linearity for the next layer. ***Fill*** kernel to fill the GPU buffer with image data (our input), weights (filters), and biases, and this is before any computations are performed. It should be noted that the ***Softmax*** kernel has very few CUDA instructions; thus, the execution time on the GPU is limited. Therefore, softmax not affected by fault injection.

C. DNNs ACCELERATORS USED IN HEALTHCARE SYSTEM

Besides the complex implementation processes involved in CNNs architecture, it is also generally accompanied by long training time. As such, due to the very high compute and

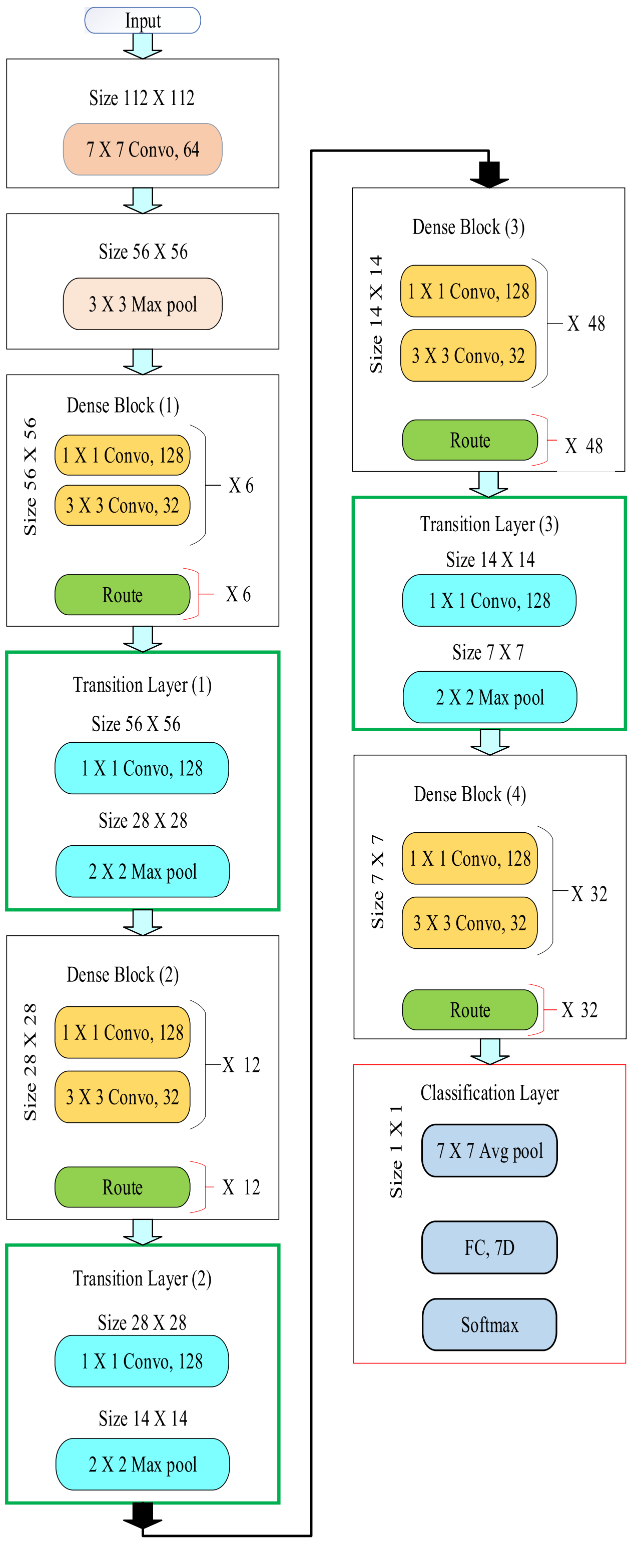


FIGURE 2. The structure of DenseNet201 after the transfer learning.

data-intensive nature of CNN, it might require quite many days or weeks to complete training involving large data sets. Furthermore, to implement CNN in healthcare applications such as screening, diagnoses, and treatment, DNN accelerators are required to execute these modes. These

TABLE 2. DenseNet201 inference kernels and their corresponding layers.

Layer	Kernel	Kernel Task
Convolutiona l	Im2col_gpu, Add_bias, Fill_gpu	the operation to matrix-multiplication operation and add biases to the necessary parameters after the matrix multiplication
Average pooling	Forward_avgpool	the kernel performs down-sampling in their corresponding layers
Max pooling	Forward_maxpool	to reduce the spatial dimension of the input volume for the next layers
Normalizatio n	Copy_gpu, Scale_bias, Normalize_gpu, Add_bias	kernel perform the normalization task for the GPU input. And dividing the input by its standard deviation to have a variance of approximately
Activation	Activation_array	apply nonlinearity to the feature maps to reduce the input linearity for the next layer
Softmax	Softmax	To calculate the probabilities of each class

DNNs accelerators are a wide range, including Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and FPGA.

Notably, the comparatively low data transfer and an enormous amount of floating-point operations in each training step make the GPU more suitable for this task [40], [9]. GPUs' most significantly desirable characteristic feature is the comparatively low cost associated with their high efficiencies towards computation. This is basically attributed to its densely parallel architecture [41]. In modern times, GPUs are generally saddled with broad-range computing responsibilities through Compute Unified Device Architecture (CUDA). Generally, the numbers of streaming multiprocessors (SM) differ between GPUs. Hence, each SM has an N number of streaming processor cores (SPs), and variables from the resident register and memories are accessible to each thread. Therefore, variables that are regularly accessed are saved in registers because registers have a large bandwidth.

As illustrated in Figure 3, it is evident that GPU has peculiar control logic, including dispatcher and internal blocks scheduler, which helps to designate the kernels. Also, the PCI-E connecter serves as a link between it and the systems' memory to enable accessible data transferring [42]. The parallelism in GPUs is accrued to the presence of SMs, each of which can complete only a single thread in a clock cycle, using registers that are designated for such tasks in the file register [43]. Therefore, there is a close relationship between GPUs' significantly high performance and the enormous on-chip parallelism, making it highly suitable for DNN algorithms such as DenseNet201. This is the main reason why NVIDIA and other manufacturers that produce gigantic GPUs often ensure deep learning by setting aside particular architectures of their GPUs to facilitate processes

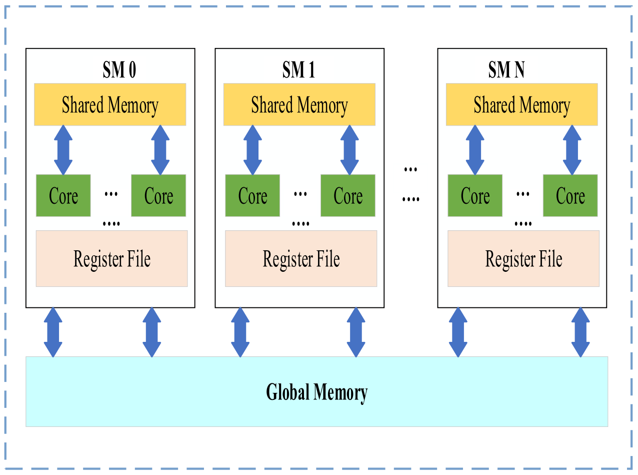


FIGURE 3. Basic GPU architecture.

such as training and inference. Consequently, the reliability of memory resources such as L1 caches, L2 caches, and register files in high-end NVIDIA GPUs are achieved by incorporating some architectural solutions designated to facilitate reliability. This is specifically achieved by utilizing a hardware approach known as Single Error Correction Double Error Detection (SECDED) ECC. However, studies on NVIDIA's GeForce revealed that the ECC feature is not supported.

D. SOFT ERRORS IN GPUs

Originally, the design of GPUs was aimed at graphics rendering. Accordingly, there was not much interest in their reliability [44]. Therefore, several reliability weaknesses were present in their architecture [13], [45], [46], [47]. Unfortunately, erroneous data can be passed on from a single corrupted thread to thousands of subsequent parallel threads, thereby resulting in a series of faults in the output. Particularly, a large number of soft errors have the capability to induce significant errors in GPU output elements [48], [44]. This will invariably result in the loss of computational integrity due to soft errors [49]. It is, however, noteworthy that in a safety-critical system (i.e., healthcare), the tolerance to failure is restricted to just 10 Failures in Time (FIT). This indicates that only one error is permitted in an operation that lapsed for 10^9 hours [50].

Soft errors in DNN accelerators such as GPU are far worse compared to other electronic devices. Therefore, they should be handled with proper attention because of two main reasons. Firstly, the improvement of latency requires a complex memory hierarchy [51]; secondly, GPUs' structure is densely parallel, which allows the easy duplication of a single fault to multiple faults [12]. Then, the generated faults can tamper with the logic operations or the data values, resulting in errors such as Silent Data Corruption (SDC). Besides, this might result in a crash or hanging of the system leading to what is known as Detected Unrecoverable Error (DUE). However, this could be masked such that there would be no observable error, resulting in what is known as Masked errors [52]. As such, the propagation of errors can proceed through different processes (herein referred to as layers) until they arrive at the program output, where they eventually trigger different problems, including object misclassification and others. Therefore, it is sufficient to infer that GPU's soft errors constitute a significant limitation in safety-critical systems and require conscious attention.

IV. DATASET AND TRAINING RESULT

To the best of our knowledge, there are just a few datasets of automated surgical instruments for public use. Most of the current datasets concentrate on the presence detection of surgical instruments derived from the M2CAI challenges. We used the m2cai16 dataset for surgical tool detection, which is one of the M2CAI challenge datasets in our training. This dataset was generated by the University Hospital of Strasbourg, France [53], [54]. It contains 20 videos of cholecystectomy procedures, split into two parts: training (15 videos) and testing (5 videos). We did not use any extra data for the training data because the data are videos with frames annotated at 1 fps. Thus, we extracted frames from the videos and took only the frames given in the annotation file. There are seven types of surgical tools in the dataset are: (a) grasper, (b) bipolar, (c) hook, (d) scissors, (e) clipper, (f) irrigator, and (g) specimen bag. Since our focus is on the model's reliability under soft errors, we applied the concept of transfer learning (TF) to train the model [55].

This is because the TF's flexibility allows the utilization of pre-trained models directly as feature extraction preprocessing and integrated into entirely new models (presence detection of surgical tools). Therefore, we adopted the pre-trained DenseNet201 model and then trained it by modifying some layers and leaving the others frozen. Without re-training the whole model from scratch. This is because features computed by the earlier layers are general and can be reused in different problem domains, while features computed by the last layers are specific and depend on the m2cai16 dataset. Hence, we performed fine-tuning to the network for surgical tool detection by choosing how much we want to adjust the 'network's weights (a frozen layer does not change during training). By adjusting hyperparameters (learning rate, epochs, batch size, and the given thresholds), we can determine the presence of the surgical tool in the image. Thus, our 'model's weights are initialized with the Densnet201 (pre-trained weights), except the classifier part, which is initialized randomly and tuned through our training.

We formulated the model learning as a multi-classification problem with Cross-Entropy as the loss function. Therefore, the network trained using stochastic gradient descent with the rate of 0.01 of learning with a momentum of 0.9 until convergence. Also, random cropping and flipping are performed for the artificial data augmentation method during the learning process. Moreover, we randomly initialize a DenseNet model at the same time to compare the training. We used the data (ten videos) for training and validations set and five videos

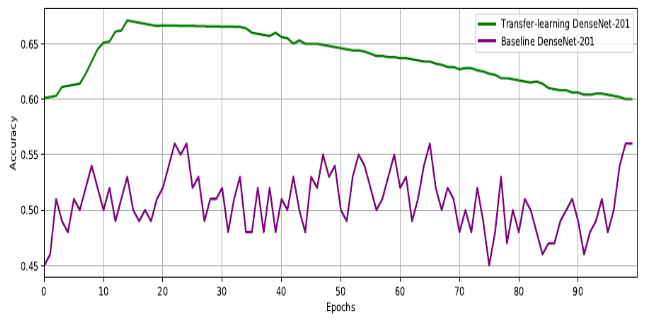


FIGURE 4. **Training accuracy of the network.**

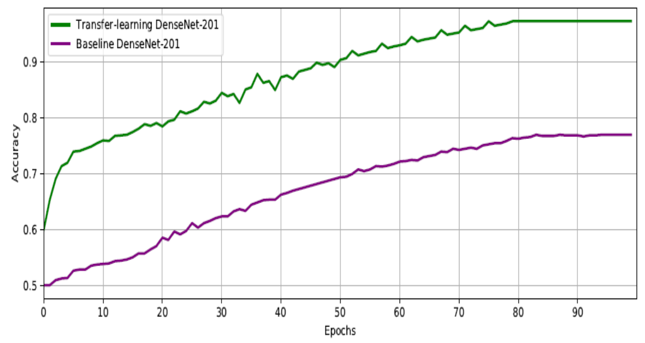


FIGURE 5. **Testing the accuracy of the network.**

for testing after resizing them into the same size (224 × 224) because the videos have different dimensions. To normalize the input, we divided each pixel by 255 to make the range of the pixels (0,1) rather than (0 to 255). Our fine-tuning was done on 100 iterations (0.01 learning rate). Figure 4 and Figure 5 show that DenseNet (Transfer-learning DenseNet-201) can detect the surgical tools with decent correctness. We reached 97% accuracy on the training set and 67% accuracy on the test set.

V. EXPERIMENTAL SETUP

Experiments were performed to evaluate the impact of soft errors in the DenseNet201 model concerning the Darknet framework. Expressly, the experiment was set up to investigate by an empirical analysis the impact of resilience error in DenseNet201 as a model that is often used in a healthcare system (e.g., presence detection of surgical tools). It is worth noting that the model was trained on the m2cai16 dataset for the surgical tool detection as described in (Section IV). The model's weight was implemented on the Darknet framework so as generate the initial accuracy before implementation on the SASSIFI environment. The GPU used in this study as the device under test has the following specifications: NVIDIA's GeForce GTX series, its microarchitecture is Maxwell, and the specific model is GTX 980. This GPU features 16 Streaming Multiprocessors (SMs), 2048 CUDA cores, and 4 GB of memory [56]. The process of fault-injection was deliberately performed using the SASSIFI fault injector to properly understand and enable accurate analysis of the occurrence and propagation of errors in the DenseNet201 model. Particularly, errors are injected by SASSIFI at the visible states of the GPU's Instruction Set Architecture (ISA). These include condition-code registers (CC), predicate registers (PR), memory values, as well as general-purpose registers (GPR). Three different modes of error-injection may be used in SASSIFI. These include Instruction Output Address (IOA), Register File (RF), and Instruction Output Value (IOV).

Nevertheless, selecting any particular injection mode depends mainly on the evaluation metric that needs to be evaluated. Two metrics were evaluated in this study. The first metric, called Architectural Vulnerability Factor (AVF), investigates the tendency of a single fault on that component to culminate in an error. This metric is generally used to investigate the reaction of applications to errors present in their memory elements. The second metric evaluated is the Program Vulnerability Factor (PVF). This investigates the tendency of modifications induced based on a particular instruction by a single fault to transform into a program output. When errors are injected in the RF mode, the register 'files' AVF is measured, whereas the PVF of the DenseNet201 program is measured when errors are injected in the IOA and IOV modes. Based on the analysis of AVF and PVF, we can easily identify the three vulnerability factors: (1) Instruction Vulnerability Factors (IVF), (2) Kernels Vulnerability Factors (KVF), and (3) Layer Vulnerability Factor (LVF).

A. FAULT MODEL USED IN HEALTHCARE APPLICATION

SASSIFI offers various bit-flip models (BFMs), and fault injection in the GPUs can be at a thread-level and in a warp (32 threads) level. Thus, transient errors that exist in memory and data paths are considered in this model. In our study, a single bit flip and random value were chosen for the DenseNet201 model injection. This is because the single bit flip is more suitable for the register file memory errors, while the random value represents the other three BFMs. Specifically, 1000 injections were performed at the RF, IOV, and IOA SASSFI modes. With this number of injections, 1.96% confidence for the worst-case statistical error bars at 95% confidence can be guaranteed. At this level, further increases in the number of injections did produce further changes in the statistics. After the injection of faults and comparisons between the program output and golden output (pure outcome), three categories could be expected, such as Masked, DUE, or SDC. However, it is worthy of note that there is a particular interest in SDC only when investigating the propagation of errors in the DenseNet201 model. This is because, after the occurrence of a crash or hangs (i.e., DUE errors), they do not further propagate to the subsequent layer. On the other hand, Masked errors are instantaneously masked once they occur. In order to gain a better understanding of the theory behind SDC errors and their mode of propagation through layers, the SDC errors have been further classified into three categories as follows:

- **Malfunction SDCs**: errors that propagate, gets to the program output, and impact the rank of objects (misclassification), due to modifications induced on the probabilities vector.
- **Light-Malfunction SDCs**: errors that propagate, get to the program output, and modify the probabilities without necessarily altering the rank of objects. This may be otherwise called tolerable SDC because it does not result in object misclassification.
- **No-Malfunction SDCs:** errors that propagate but could not arrive at the final program output because it has been masked at a particular layer. It should be noted that this type of error differs from Masked errors because masked errors do not propagate at all.

VI. RESULTS AND ANALYSIS

In this section, we present our findings and analyze them. To evaluate the resilience of the DenseNet201 model, we conduct a detailed model's sensitivity analyses from several perspectives by evaluating two evaluation metrics as follows; (1) Layers vulnerability analysis (LVF) and (2) Kernels vulnerability analysis (KVF).

A. LAYER VULNERABILITY ANALYSIS

In this subsection, we analyze the error propagation in DenseNet201 through its different layers. As described in (Section III-B) based on the Darknet framework, DenseNet201 utilizes a composite function containing batch normalization and ReLU after a 3×3 convolution layer. The composite function output is concatenated with the input then passed to the following composite function, as seen in Figure 2. The composite function begins with a bottleneck 1×1 convolution layer with 128 filters. This is done to reduce the input feature-maps and make the larger convolution more efficient. This is again followed by batch normalization and a ReLU activation with the 3×3 convolution layer containing only 32 filters. Because of the filter concatenation across the network, each composite function needs only to perform a small piece. The network performs a specific amount of composite functions in a dense block then uses a transition layer to compress the network. This compression begins with a 1×1 convolution to reduce the filter dimensionality, then a 2×2 max pool with a stride of two to halve the output size.

Figure 6 to Figure 8 shows the AVF values for injections in the RF site and PVF values for injections in IOA and IOV sites, respectively. We investigate and calculate the three SDC categories for each layer, as explained in (Section V-A). Thus, we measure AVF and PVF values for the given errors to identify the layers likely to produce errors that crucially change the model's prediction (i.e., object misclassification). In Figure 6, injecting faults in RF mode, layers tend to produce big amounts (37.50%) of DUEs. It produces a small amount of 5.20% of Light-Malfunction SDCs and produces 0.70% of Malfunction SDCs; this shows that RF injections do not significantly impact the layer's resilience against SDC errors. However, layers producing No-Malfunction SDCs about 56.60% of the SDC errors that have been injected. Whereas injecting faults into instruction (IOA and IOV) layers produces large amounts of SDC (Malfunction and Light-Malfunction) errors, which affects the model's resilience. In Figure 7 IOA, layers generate a small amount of the Malfunction SDCs errors and a large amount of Light-Malfunction errors in the 2.20% and 19.15%, respectively, and most of the injected errors 66.15% No-Malfunction SDCs. However, layers like (i.e., 0, 1, 2, 6, 8,12, 17, 20, 21, 22, 50, 63, 123, 163) produce Malfunction on the average 0.2% SDCs errors, which is high in safety-critical application (presence detection of surgical tools). Nevertheless, most of the layers produce 12.50% DUE errors.

In Figure 8 IOV, layers generated Malfunction SDCs on 16.49%, a high percentage in the safety-critical application. Statistically, layers 3, 6, 9, 12, 15, and 18 represent the first DenseNet201 block. These layers have the same behaviors (i.e., filters 32) and contribute 2.8% of the overall Malfunction SDCs errors. On the other hand, layers 2, 5, 8, 11, 14, and 17 also represented the first DenseNet201 block and have the same behaviors (i.e., filters 128) and contribute with the highest percentage of Malfunction SDCs 3.9%. Compare these layers to previous layers on the Malfunction amount, and the main reason is because of the image size and the number of filters. All these layers (2, 3, 5, 6, 8, 9, 11, 12, 14, 15, 17, and 18) produce a high amount of Light-Malfunction and No-Malfunction with a percentage on an average of 0.6% and 1.4%, respectively. However, the IOV mode still produces DUEs errors on an average of 0.05%; nevertheless, Route layers (R) are not affected at all with error injections, and the reason is that R layers only take the results of the preceding layers without any processing. As we can see in Figure 6, Figure 7, and Figure 8 (RF, IOA & IOV), the model starting with a high number of errors in the first layers and decrease until the end of the model layers, and the reason because of the image size in the first layer 256×256 and starts reducing until the last layer in the model. Thus, it implies why the error rate is very high at the beginning of the model layers.

B. KERNEL VULNERABILITY ANALYSIS

In this subsection, we analyze the resilience of the DenseNet201 model through the kernel's perspective. The part of the source code that is implemented on the GPU is called a kernel. In order to demonstrate how different kernels have different vulnerabilities and how they eventually contribute to the program output (i.e., object classification), we have collected all the static kernels that are needed for inference (training kernels are not included) that are involved in executing DenseNet201 model on a GPU. After the faults are injected, SASSIFI provides the capability of obtaining kernels' details of the injected program. As Table 2 shows, ten kernels are used to implement DenseNet201's pre-trained model on the Darknet framework.

Figure 9 to Figure 11 shows the KVF for DenseNet201 modes: RF, IOA, and IOV. As introduced by [22], the KVF means the probability of faults in a kernel to affect the

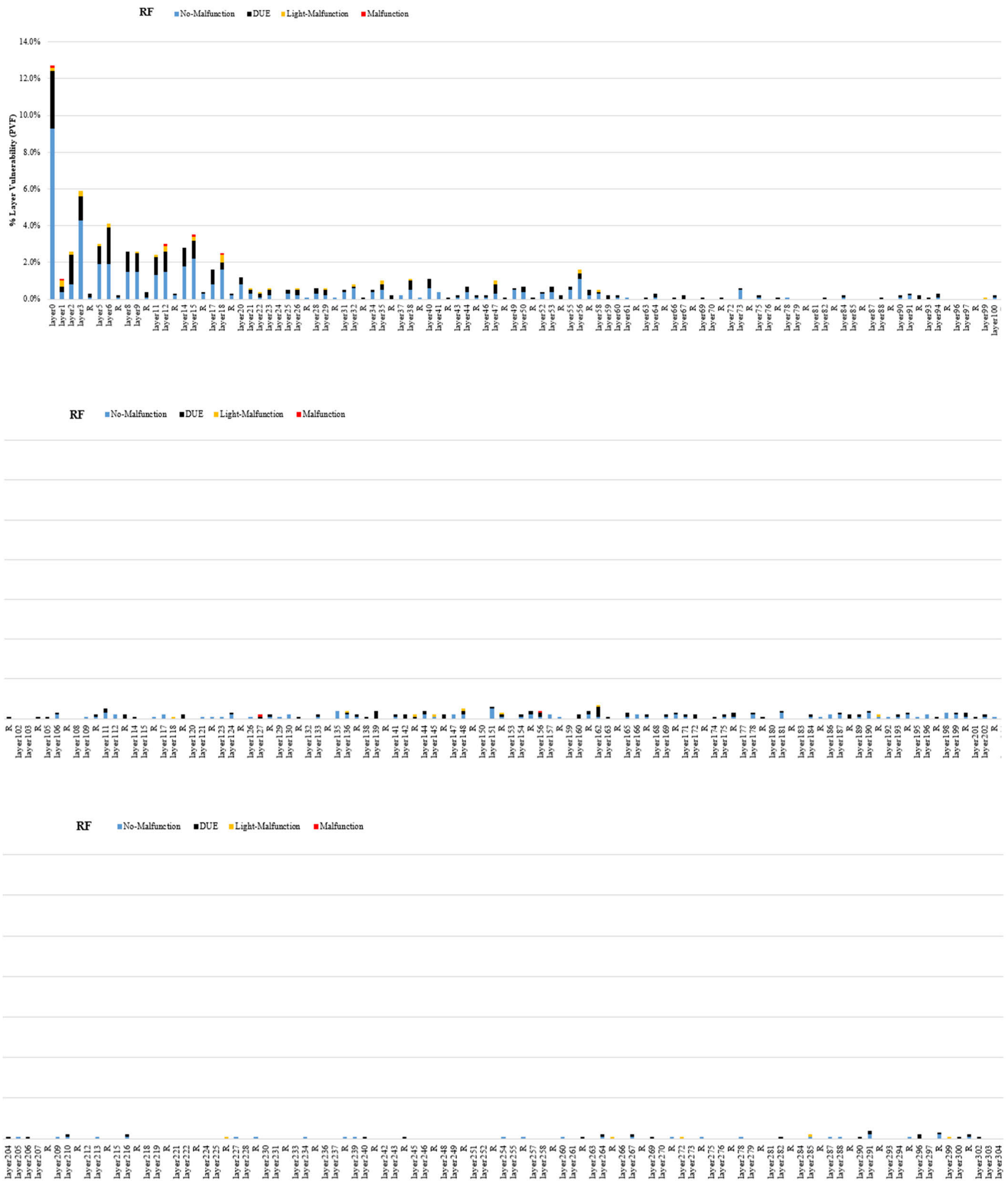


FIGURE 6. AVF of each layer for no-malfunction SDCs, light-malfunction SDCs, and malfunction SDCs DenseNet201 (RF).

model's computations. It is worth noting that in all Figures, the probabilities of the whole graph sum up to 100% (not each kernel's vertical bars) because every DenseNet201 model's program consists of all these kernels. In other words, DenseNet201 programs are divided up into small pieces of programs (kernels) that are executed in the GPU. By looking

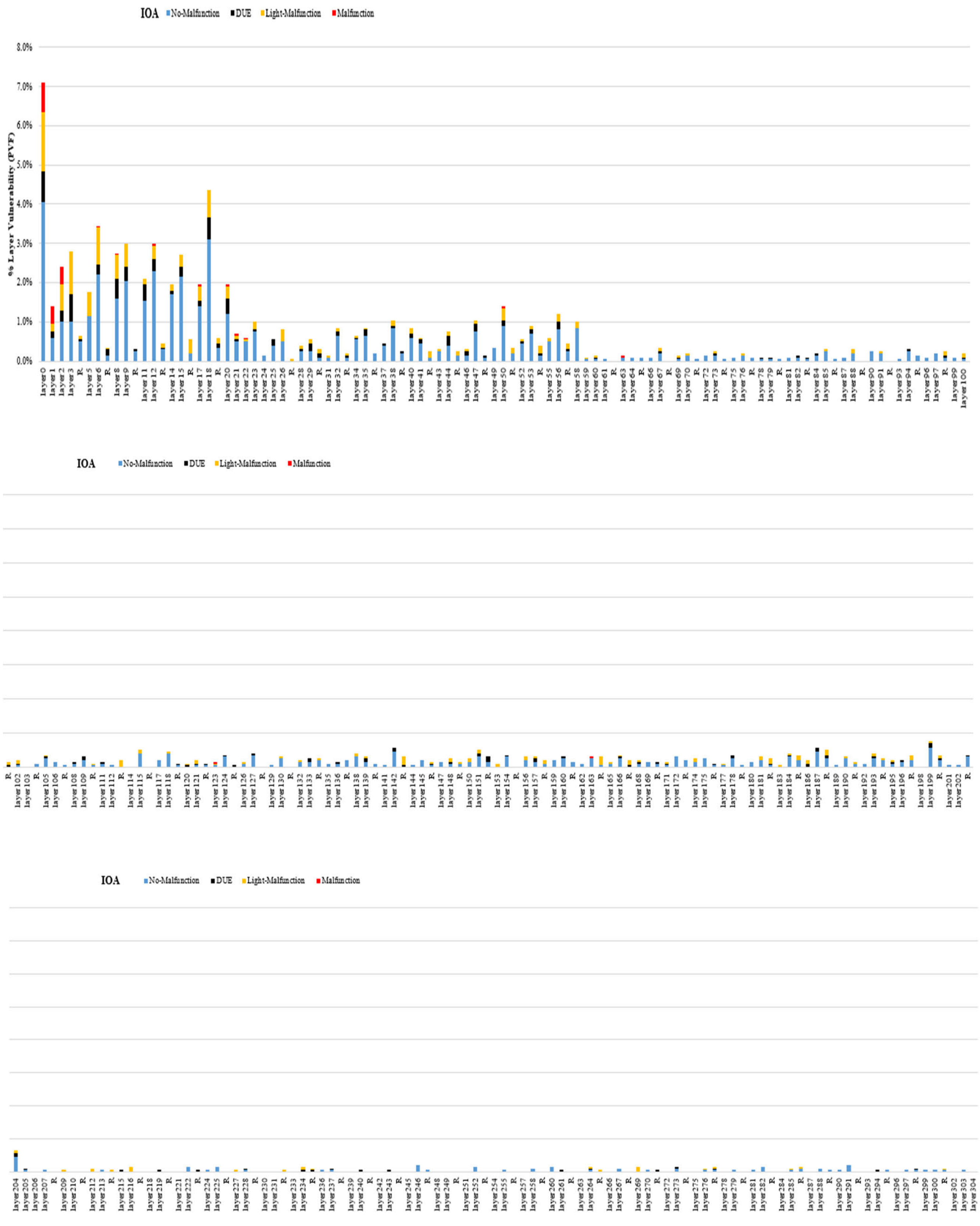


FIGURE 7. PVF of each layer for no-malfunction SDCs, light-malfunction SDCs, and malfunction SDCs DenseNet201 (IOA).

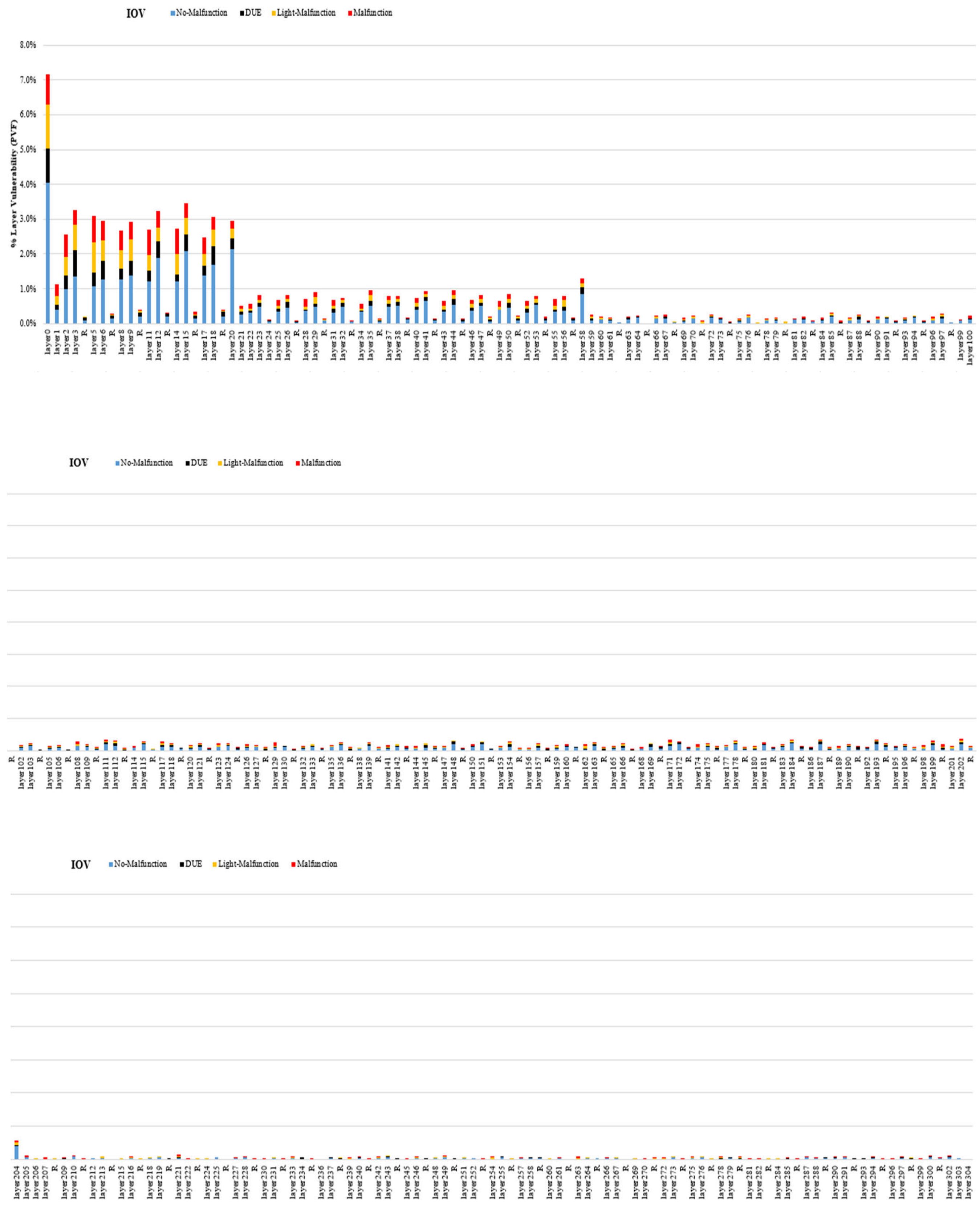


FIGURE 8. PVF of each layer for no-malfunction SDCs, light-malfunction SDCs, and malfunction SDCs DenseNet201 (IOV).

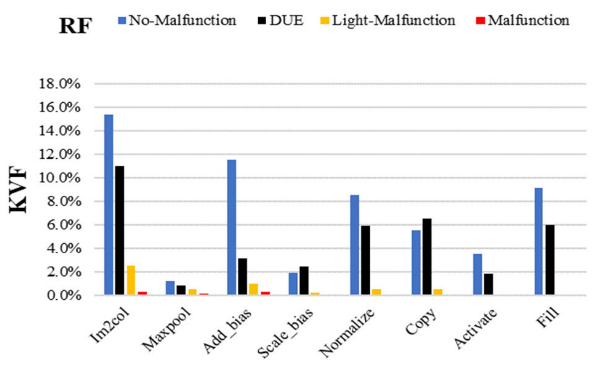


FIGURE 9. Kernels vulnerability of DenseNet201 models for RF mode.

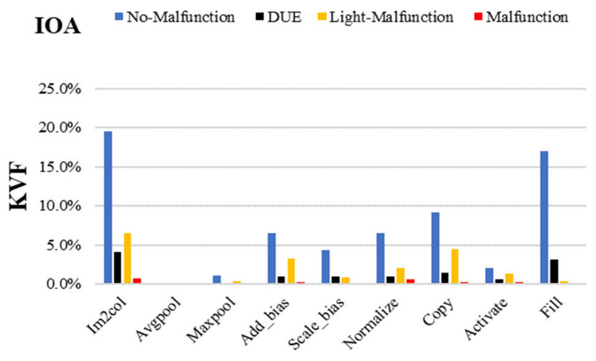


FIGURE 10. Kernels vulnerability of DenseNet201 models for IOA mode.

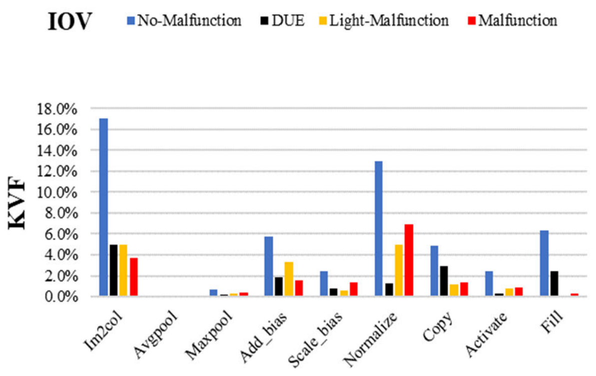


FIGURE 11. Kernels vulnerability of DenseNet201 models for IOV mode.

at Figure 9 to Figure 11, we identify which kernel produces Malfunction, Light-Malfunction, No-Malfunction, and DUE errors more than others do and which one is resilient. Overall, all kernels produce errors in the models.

Moreover, we find that the top-4 vulnerable kernels for the DenseNet201 model are *Im2col*, *Add_bias*, *Normalize,* and *Copy* in all three modes (RF, IOA, and IOV). Other kernels produce a small number of DUEs and/or SDC in all. Thus, they have very high resilience to soft errors. Besides, the *Fill* kernel produces minor SDC errors, and thus, they have high resilience to soft errors. Furthermore, we can notice that more than one kernel can contribute to building one CNN layer (e.g., Conv. or activation function layers). From a different point of view, this makes it statistically to determine which layer is more vulnerable to Malfunction and Light-Malfunction and then make a better decision in mitigating errors. As in Figure 9, kernels produce more DUE errors in the RF model than SDC errors in DenseNet201, whereas in IOA and IOV modes (Figure 10 and Figure 11), kernels more likely to generate SDC errors more than DUE errors. The reason, as stated by [57], is that injecting at the RF site is considered the lowest level of injection, while injecting at IOA and IOV sites is performed at a higher level since we manipulate instructions. To conclude, static kernels of the DenseNet201 models have different vulnerabilities, and our results help identify which kernels to be duplicated for a cost-effective solution instead of duplicating the whole model, which is a costly technique.

VII. SELECTIVE MITIGATION TECHNIQUE FOR HEALTHCARE APPLICATION

Based on the previous analysis in Section (VI), we propose a selective mitigation technique for the Malfunction and Light-Malfunction errors in the various kernels. It should be noted that this technique is based on the Triple Modular Redundancy (TMR) to mitigate only the kernels that produce the Malfunction and Light-Malfunction SDCs.

A. LAYER VULNERABILITY EVALUATION

This subsection discusses the selective mitigation technique that we use to mitigate the soft errors in our health-care application. Nevertheless, note that it is application-specific, and it depends on the DNN accelerates and DNNs model (i.e., network topology, layer positions, and kernels). Therefore, based on our in-depth analysis, we provided in Section (VI-A and VI-B) for the layers analysis and kernels analysis after injecting DenseNet201, which is often used in healthcare applications, such as presence detection of surgical tools. It is always essential that healthcare models have high reliability because minor errors may lead to injury or death. Therefore, when we injected our model as in Figure 6 to Figure 8 for RF, IOA, and IOV show the layers' vulnerability. We calculate the Malfunction SDCs, Light-Malfunction SDCs, No-Malfunction SDCs, and DUEs.

However, Figure 12 shows RF mode after applying our mitigation technique; the experimental result shows only 0.1 % Malfunction SDCs and Light-Malfunction SDCs produced, while it still produced a significant amount of the No-Malfunction SDCs in the percentage 62.40 %. Despite the amount of the DUEs still high as 37.40 %. On the anther hand, Figure 13 and Figure 14 shows IOA and IOV, both of the modes IOA and IOV produced less amount of the DUEs 13.3% and 15.8%, respectively, compared to the RF mode, and the reason that IOA and IOV have a different level of injections. Therefore, in Figure 13, IOA

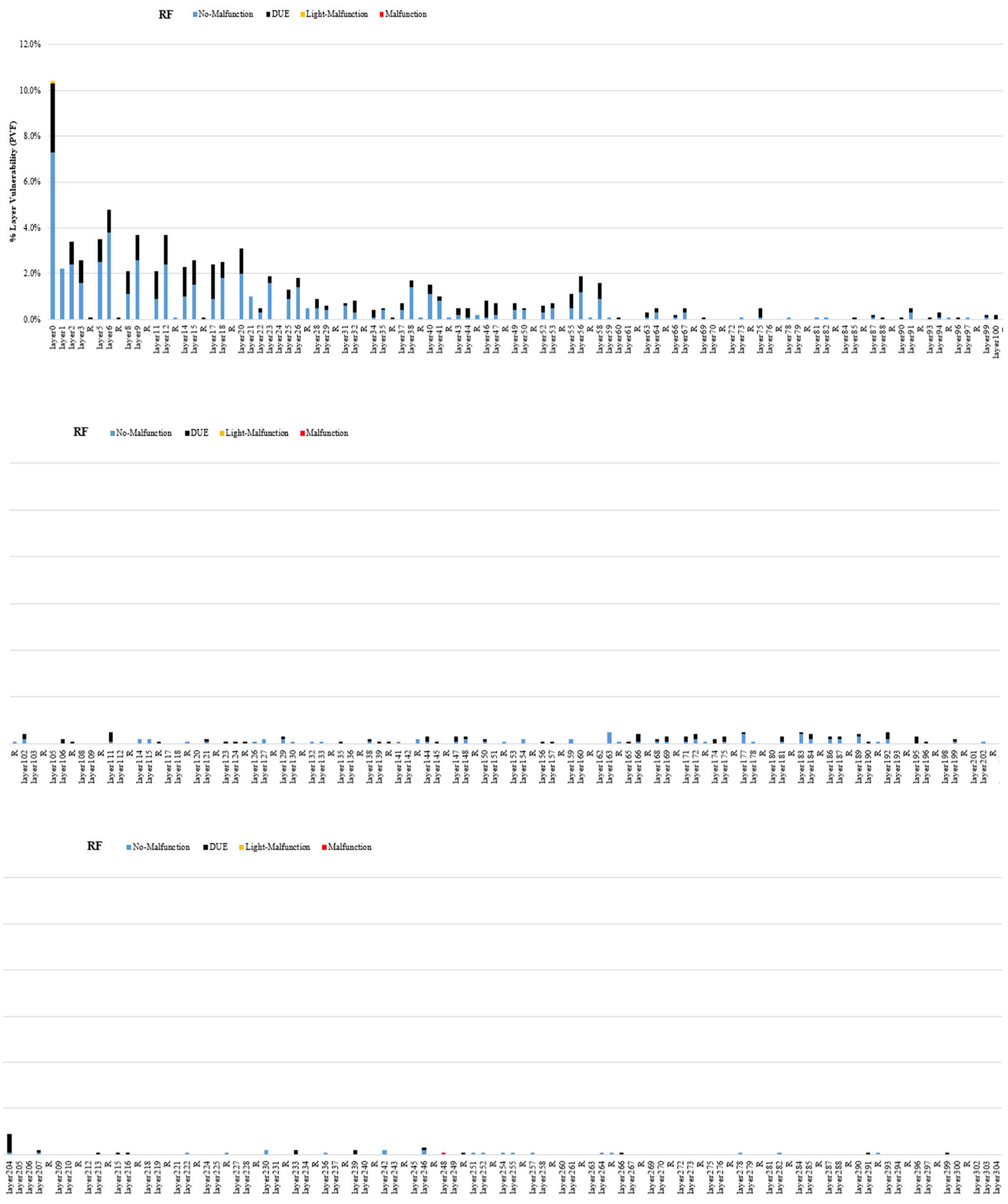


FIGURE 12. Evaluation for each layer of DenseNet201 model (after applying mitigation technique) in RF mode.

produced only 0.1 % of the Malfunction SDCs and 1.4% of the Light-Malfunction SDCs. Meanwhile, most of the errors of 85.2% No-Malfunction SDCs. On the anther hand, IOV in Figure 14 produced less amount of Malfunction SDCs and Light-Malfunction SDCs 0.5% and 0.4%, respectively. Where 83.2% of the errors No-Malfunction.

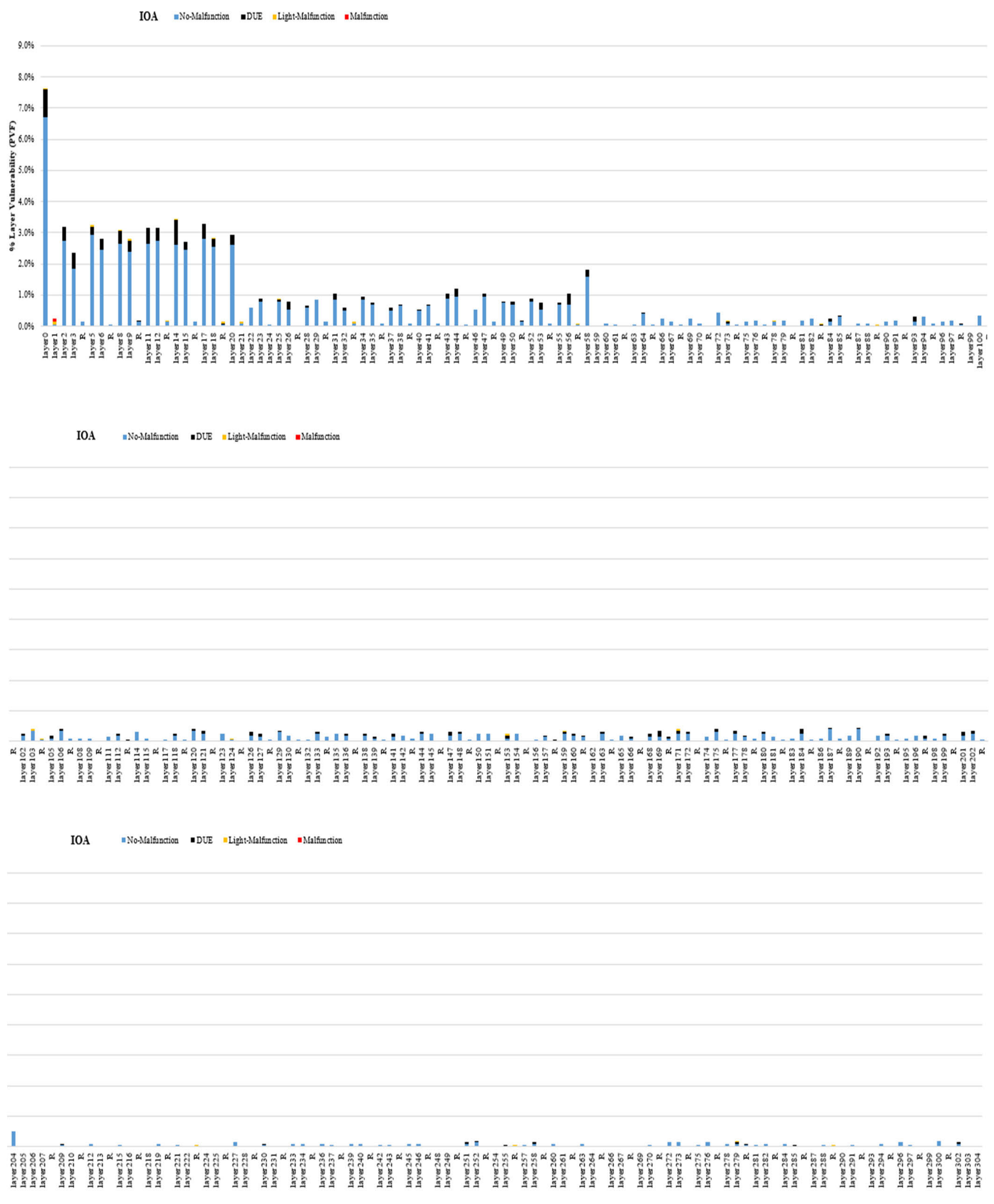


FIGURE 13. Evaluation for each layer of DenseNet201 model (after applying mitigation technique) in IOA mode.

B. KERNEL EVALUATION

In this subsection, the RF, IOA, and IOV modes in Figures 9, 10, and 11 are evaluated for kernels by applying our mitigation technique. Based on our analysis in Section (VI-B), the top-4 vulnerable kernels for DenseNet201 are *Im2col*, *Add_bias*, *Normalize,* and *Copy*

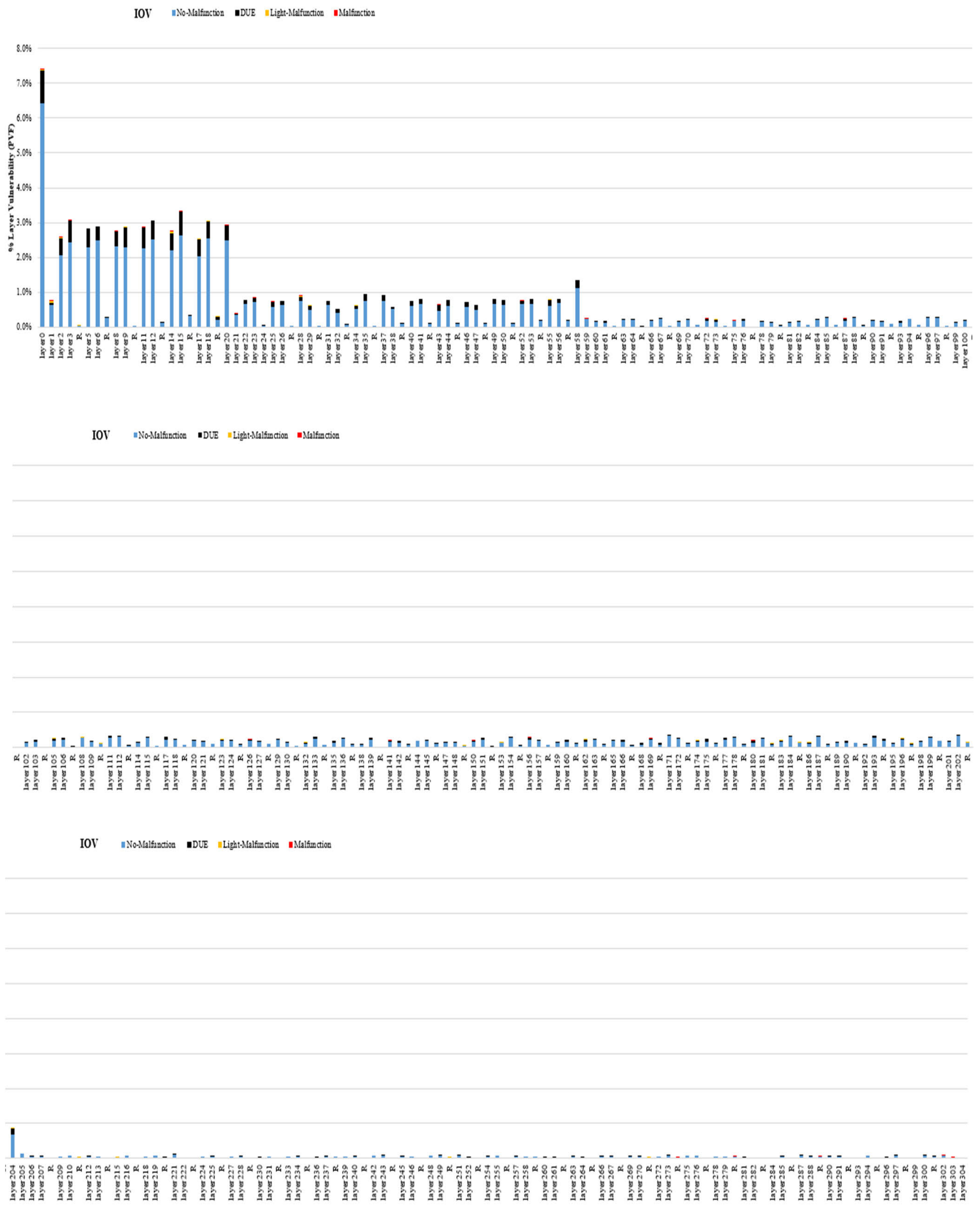


FIGURE 14. Evaluation for each layer of DenseNet201 model (after applying mitigation technique) in IOV mode.

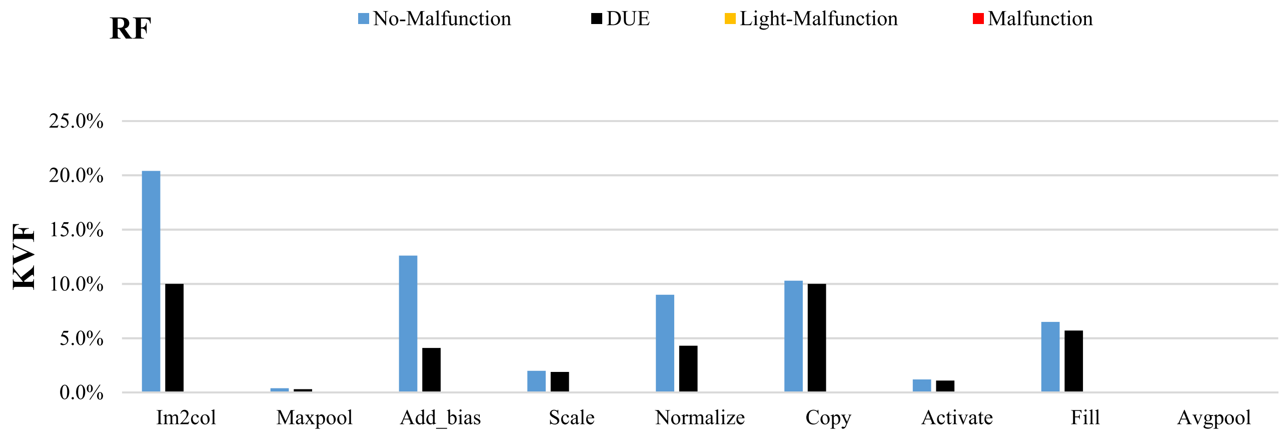


FIGURE 15. Kernels vulnerability of DenseNet201 models for RF mode (after applying mitigation technique).

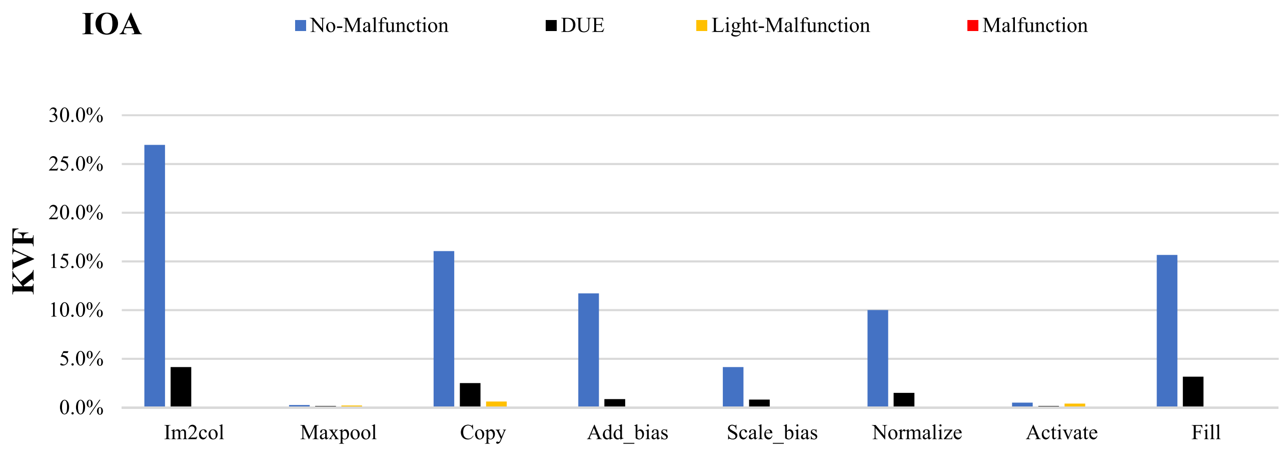


FIGURE 16. Kernels vulnerability of DenseNet201 models for IOA mode (after applying mitigation technique).

TABLE 3. Kernels injection.

Injection	RF				IOA				IOV			
kernel	No-M	DUE	Li-M	M	No-M	DUE	Li-M	M	No-M	DUE	Li-M	M
Im2col	15.4%	11.0%	**2.5%**	**0.3%**	19.5%	4.2%	**6.5%**	**0.7%**	17.1%	5.0%	**5.0%**	**3.7%**
Maxpool	1.2%	0.8%	0.5%	0.1%	1.1%	0.1%	0.4%	0.2%	0.7%	0.2%	0.3%	0.4%
Add_bias	11.5%	3.1%	**1.0%**	**0.3%**	6.6%	1.0%	**3.2%**	**0.3%**	5.7%	1.9%	**3.3%**	**1.6%**
Scale_bias	1.9%	2.4%	0.2%	0.0%	4.4%	1.0%	0.8%	0.0%	2.4%	0.8%	0.6%	1.4%
Normalize	8.5%	5.9%	**0.5%**	**0.0%**	6.5%	1.0%	**2.1%**	**0.6%**	13.0%	1.3%	**5.0%**	**6.9%**
Copy	5.5%	6.5%	**0.5%**	**0.0%**	9.2%	1.5%	**4.5%**	**0.3%**	4.9%	2.9%	**1.2%**	**1.4%**
Activate	3.5%	1.8%	0.0%	0.0%	2.0%	0.7%	1.4%	0.2%	2.5%	0.3%	0.8%	0.8%
Fill	9.1%	6.0%	0.0%	0.0%	17.0%	3.2%	0.4%	0.1%	6.3%	2.5%	0.0%	0.3%
Avgpool	0.00%	0.00%	0.00%	0.00%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

in all three modes. Figures 15, 16, and 17 show that all the *Malfunction SDCs* in these kernels become *No-Malfunction* as we summarize in Table 3 and Table 4 the percentage of the *Malfunction, Light-Malfunction,* and *No-Malfunction SDCs* for each kernel before and after we injected all the kernels and applied our mitigation technique. Our mitigation technique

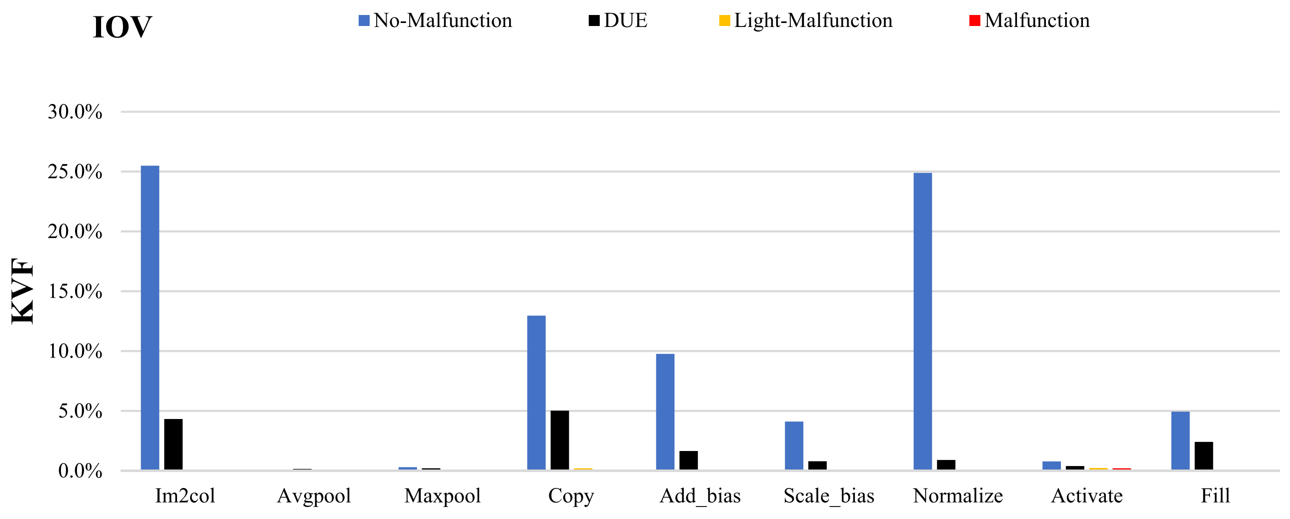


FIGURE 17. Kernels vulnerability of DenseNet201 models for IOV mode (after applying mitigation technique).

TABLE 4. Kernels after implementing our mitigation technique.

Mitigation	**RF**				**IOA**				**IOV**			
kernel	No-M	DUE	Li-M	M	No-M	DUE	Li-M	M	No-M	DUE	Li-M	M
Im2col	20.4%	10.0%	**0.0%**	**0.0%**	27.0%	4.2%	**0.0%**	**0.0%**	25.5%	4.3%	**0.0%**	**0.0%**
Maxpool	0.4%	0.3%	0.0%	0.0%	0.3%	0.2%	0.2%	0.1%	0.3%	0.2%	0.1%	0.1%
Add_bias	12.6%	4.1%	**0.0%**	**0.0%**	11.7%	0.9%	**0.0%**	**0.0%**	9.8%	1.7%	**0.0%**	**0.0%**
Scale_bias	2.0%	1.9%	0.0%	0.0%	4.2%	0.8%	0.1%	0.0%	4.1%	0.8%	0.0%	0.0%
Normalize	9.0%	4.3%	**0.0%**	**0.0%**	10.0%	1.5%	**0.0%**	**0.0%**	24.9%	0.9%	**0.0%**	**0.0%**
Copy	10.3%	10.0%	**0.1%**	**0.1%**	16.1%	2.5%	**0.6%**	**0.0%**	13.0%	5.0%	**0.2%**	**0.0%**
Activate	1.2%	1.1%	0.0%	0.0%	0.5%	0.2%	0.4%	0.0%	0.8%	0.4%	0.2%	0.2%
Fill	6.5%	5.7%	0.0%	0.0%	15.7%	3.2%	0.1%	0.0%	4.9%	2.4%	0.0%	0.1%
Avgpool	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%

shows significant improvement in the DenseNet201 model, as we show in Table 4. The errors in the top-4 vulnerable kernels are *Im2col*, *Add_bias*, *Normalize,* and *Copy.* Errors in these kernels have been reduced from 4.50% to 0.10% in *Light- Malfunction SDCs* of the RF mode and from 16.30% to 0.60% and 1.8% to 0.00% in *Light-Malfunction* and *Malfunction* errors, respectively, for IOA mode. Meanwhile, they have been significantly improved in IOV mode, where the *Light-Malfunction* errors have been reduced from 14.40% to 0.20% and 13.60% to 0.00% *Malfunction.*

C. PERFORMANCE OVERHEADS EVOLUTION

This section evaluates our proposed solution by measuring the performance overheads for the whole model (DenseNet) and vulnerable kernels. That is by calculating the execution time (performance) for the whole model and the vulnerable kernels before and after implementing our mitigation technique. Table 5 shows that the model performance (exaction time) has only increased by **0.3035%** and improved model's reliability. Our technique shows high significant

TABLE 5. Performance overhead evolution for the whole model and vulnerable kernels before and after protecting.

	Number of invocations	Unprotected (time by ms)	Protected (time by ms)
Im2col	99	0.000689	0.031239
Add_bias	201	0.002201	0.010563
Normalize	200	0.004849	0.0130992
The whole model	1	**1.996011**	**2.0020688**

error mitigation, as shown in Table 4, by only protecting a few kernels. Hence, by leaving the non-vulnerable kernels without covering them by our technique, the unnecessary overhead has been removed. The reason is that because of safety-critical systems (healthcare applications) that come with strict deadlines do not bear the overhead associated with the protection of a whole model.

VIII. COMPARISON OVERHEADS OF OUR SELECTIVE MITIGATION TECHNIQUE TO ABFT, DMR, AND TMR

In Section (VII – C), we have assessed our proposed technique's performance costs by calculating the execution time

TABLE 6. Comparison of the overhead of unhardened model, selective mitigation technique, DMR, and TMR.

Kernels (time by ms)	Number of invocations	Unhardened	SMT	ABFT	DMR	TMR
Im2col	99	0.000689	0.031239	0.0813729	0.001379	0.031305
Add_bias	201	0.002201	0.010563	0.0606889	0.004252	0.0111748
Normalize	200	0.004849	0.0130992	0.004838	0.009677	0.01499
Maxpool	4	0.07114	0.07109	0.1713729	0.14234	0.2134
Avgpool	1	0.03546	0.0329	0.06836	0.07112	0.10662
Copy	200	0.5334011	0.5341002	1.0667999	1.0668002	1.6002013
Scale	200	0.4978501	0.4979401	0.9957902	0.9956802	1.4935203
Activation	201	0.29007	0.2844701	0.3847339	0.5689612	0.8534403
Fill	201	0.4221108	0.3854272	0.4354541	0.8534402	1.2801603
Softmax	1	0.13824	0.14124	0.1923669	0.28448	0.42672
The whole model	1	**1.996011**	**2.0020688**	**3.4617777**	**3.9981298**	**6.031532**

SMT: Selective Mitigation Technique
ABFT: Algorithm-Based Fault Tolerance
DMR: Double Modular Redundancy
TMR: Triple Modular Redundancy

for the whole model (DenseNet) and only vulnerable kernels. Here, we compare our Selective Mitigation Technique with three existing techniques: ABFT, DMR, and TMR, to further investigate their performance penalties on DenseNet architecture. Table 6 summarizes this comparison. It is worth noting that ABFT does not protect different kernels, as our technique does. Instead, it only protects matrix multiplication-related operations. Nevertheless, since most of the kernels include these operations, their execution times will be affected by the ABFT solution. Therefore, we can calculate each kernel's execution time before and after adopting the ABFT technique. DMR and TMR, on the other hand, are straightforward to compute their overheads, which is by hardening all the eleven kernels of our model (DensNet201), duplicating and replicating, respectively. By looking at Table 6, we can see that our technique (SMT) achieves the lowest overhead with sufficient reliability. This is because SMT selectively hardened only the vulnerable kernels, as listed in Table 5. By exploiting the selective mitigation technique, which heavily depends on the in-depth analysis carried on in Section VI, the overhead can be notably reduced. Consequently, the overhead has only increased by 0.3035%, compared to the ABFT, DMR, and TMR, where their overheads increased by 73.435%, 100.306%, and 202.179%, respectively. Therefore, our technique shows high significant error mitigation while removing the unnecessary overheads in safety-critical systems (healthcare applications), which could be a serious issue.

IX. CONCLUSION

As human life is involved, healthcare applications (i.e., presence detection of surgical tool) are safety-critical systems. Thus, they should be highly reliable because small errors might lead to serious injury or death.As discussed in Section (III-B), DNNs have different behavior and workflows because of various DNNs architectures. Therefore, the reliability of DNN models depends on many factors, such as the network topology, layers positions, and layers number. Thus, it is difficult to directly apply a specific DNN's case to solve other architectures.

In this paper, we analyzed the DenseNet201 model trained on a dataset of presence detection of the surgical tool and evaluated the impact of Malfunction, Light-Malfunction, and No-Malfunction SDCs of the soft errors on the reliability of the DenseNet201 on GPUs. We proposed a mitigation technique to reduce the Malfunction and Light-Malfunction in three modes. Our technique shows a high reduction rates of errors, in the top-4 vulnerable kernels in RF mode from 2.5% to 0.0% *Im2col*, 1.0% to 0.0% *Add_bias*, 0.5% to 0.0% *Normalize,* and 0.5% to 0.1% *Copy* in *Light- Malfunction*. Whereas, the errors in IOA have been reduced from 6.5% to 0.0% *Im2col*, 3.2% to 0.0% *Add_bias*, 2.1% to 0.0% *Normalize,* and 4.5% to 0.6% *Copy* in *Light- Malfunction.* Meanwhile, they have been significantly improved in IOV from 3.7% to 0.0% *Im2col*, 1.6% to 0.0% *Add_bias*, 6.9% to 0.0% *Normalize,* and 1.4% to 0.0% *Copy* in *Malfunction*. Besides, our mitigation technique achieved high rates of decrease for No-Malfunction SDCs, from **56.60%, 66.15%,** and **52.59%** to **62.40%, 85.2%,** and **83.20%** in the RF, IOA, and IOV modes, respectively. It is worth mentioning that, although the Copy kernel generates a small amount of Light-Malfunction SDCs in all three modes and a tiny amount of Malfunction SDCs in RF, we did not harden it. The reason is that, as most of the layers call the Copy kernel to bring the input, hardening it would increase the model's overhead. Finally, we have compared our technique's performance overhead with three well-known protection techniques: ABFT, DMR, and TMR. The proposed solution demonstrates its efficiency, showing the lost overhead compared to others while correcting up 84.8% of the SDCs.

REFERENCES

[1] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech and time series," in *The Handbook of Brain Theory and Neural Networks*, vol. 3361. Cambridge, MA, USA: MIT Press, 1995, pp. 255–258. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf

[2] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4438–4446.

[3] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-Net: Fully convolutional neural networks for volumetric medical image segmentation," in *Proc. 4th Int. Conf. 3D Vis. (3DV)*, Oct. 2016, pp. 565–571.

[4] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 354–370.

[5] S. Wang, A. Raju, and J. Huang, "Deep learning based multi-label classification for surgical tool presence detection in laparoscopic videos," in *Proc. IEEE 14th Int. Symp. Biomed. Imag. (ISBI)*, Apr. 2017, pp. 620–623, doi: 10.1109/ISBI.2017.7950597.

[6] S. Lu, Z. Lu, and Y.-D. Zhang, "Pathological brain detection based on AlexNet and transfer learning," *J. Comput. Sci.*, vol. 30, pp. 41–47, Jan. 2019, doi: 10.1016/j.jocs.2018.11.008.

[7] E. Rajih, C. Tholomier, B. Cormier, V. Samoulian, T. Warkus, M. Liberman, H. Widmer, J.-B. Lattouf, A. M. Alenizi, M. Meskawi, R. Valdivieso, P.-A. Hueber, P. I. Karakewicz, A. El-Hakim, and K. C. Zorn, "Error reporting from the da vinci surgical system in robotic surgery: A Canadian multispecialty experience at a single academic centre," *Can. Urol. Assoc. J.*, vol. 11, no. 5, p. 197, May 2017, doi: 10.5489/cuaj.4116.

[8] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in Deep Learning Neural Network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Denver, CO, USA, 2017, pp. 1–12, doi: 10.1145/3126908.3126964.

[9] T. Kalaiselvi, P. Sriramakrishnan, and K. Somasundaram, "Survey of using GPU CUDA programming model in medical image analysis," *Informat. Med. Unlocked*, vol. 9, pp. 133–144, Aug. 2017.

[10] A. A. Shvets, A. Rakhlin, A. A. Kalinin, and V. I. Iglovikov, "Automatic instrument segmentation in robot-assisted surgery using deep learning," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2018, pp. 624–628.

[11] E. Kuznetsov and V. Stegailov, "Porting CUDA-based molecular dynamics algorithms to AMD ROCm platform using HIP framework: Performance analysis," in *Proc. Russian Supercomputing Days*, 2019, pp. 121–130.

[12] Z. You *et al.*, "White Paper on AI chip technologies 2018," Beijing Innov. Center Future Chips (ICFC), Beijing, China, Tech. Rep., 2018.

[13] D. A. G. Oliveira, L. L. Pilla, T. Santini, and P. Rech, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 791–804, Mar. 2016, doi: 10.1109/TC.2015.2444855.

[14] H. Alemzadeh, J. Raman, N. Leveson, Z. Kalbarczyk, and R. K. Iyer, "Adverse events in robotic surgery: A retrospective study of 14 years of FDA data," *PLoS ONE*, vol. 11, no. 4, pp. 1–20, 2016, doi: 10.1371/journal.pone.0151470.

[15] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen, "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19490–19503, 2020, doi: 10.1109/ACCESS.2020.2968129.

[16] F. F. D. Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Trans. Rel.*, vol. 68, no. 2, pp. 663–677, Jun. 2019.

[17] F. F. D. Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, "Evaluation and mitigation of soft-errors in neural network-based object detection in three GPU architectures," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2017, pp. 169–176, doi: 10.1109/DSN-W.2017.47.

[18] C. Lunardi, F. Previlon, D. Kaeli, and P. Rech, "On the efficacy of ECC and the benefits of FinFET transistor layout for GPU reliability," *IEEE Trans. Nucl. Sci.*, vol. 65, no. 8, pp. 1843–1850, Aug. 2018.

[19] C. S. Vidya and B. P. V. Kumar, "Reliability analysis in healthcare imaging applications," *Indian J. Sci. Technol.*, vol. 9, p. 34, Sep. 2016, doi: 10.17485/ijst/2016/v9i34/100988.

[20] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman, "Analysis of safety-critical computer failures in medical devices," *IEEE Secur. Privacy*, vol. 11, no. 4, pp. 14–26, Jul. 2013, doi: 10.1109/MSP.2013.49.

[21] B. Zhang, S. Wang, L. Dong, and P. Chen, "Surgical tools detection based on modulated anchoring network in laparoscopic videos," *IEEE Access*, vol. 8, pp. 23748–23758, 2020, doi: 10.1109/ACCESS.2020.2969885.

[22] F. F. D. Santos, L. Carro, and P. Rech, "Kernel and layer vulnerability factor to evaluate object detection reliability in GPUs," *IET Comput. Digit. Techn.*, vol. 13, no. 3, pp. 178–186, May 2019.

[23] M. Islam, D. A. Atputharuban, R. Ramesh, and H. Ren, "Real-time instrument segmentation in robotic surgery using auxiliary supervised deep adversarial learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2188–2195, Apr. 2019, doi: 10.1109/LRA.2019.2900854.

[24] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[25] G. Li, K. Pattabiraman, C. Y. Cher, and P. Bose, "Understanding error propagation in GPGPU applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Salt Lake City, UT, USA, Nov. 2016, pp. 240–251, doi: 10.1109/SC.2016.20.

[26] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: Review, opportunities and challenges," *Briefings Bioinf.*, vol. 19, no. 6, pp. 1236–1246, Nov. 2018, doi: 10.1093/bib/bbx044.

[27] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature Med.*, vol. 25, no. 1, pp. 24–29, Jan. 2019.

[28] Z. Liang, G. Zhang, J. X. Huang, and Q. V. Hu, "Deep learning for healthcare decision making with EMRs," in *Proc. IEEE Int. Conf. Bioinf. Biomed. (BIBM)*, Nov. 2014, pp. 556–559.

[29] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Predicting healthcare trajectories from medical records: A deep learning approach," *J. Biomed. Informat.*, vol. 69, pp. 218–229, May 2017.

[30] O. Faust, Y. Hagiwara, T. J. Hong, O. S. Lih, and U. R. Acharya, "Deep learning for healthcare applications based on physiological signals: A review," *Comput. Methods Programs Biomed.*, vol. 161, pp. 1–13, Jul. 2018.

[31] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, pp. 5455–5516, Apr. 2020.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–14.

[34] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1–9, doi: 10.1109/CVPR.2015.7298594.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 2, pp. 84–90, Jun. 2012, doi: 10.1145/3065386.

[36] G. Huang, Z. Liu, G. Pleiss, L. Van Der Maaten, and K. Weinberger, "Convolutional networks with dense connectivity," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, May 23, 2019, doi: 10.1109/tpami.2019.2918284.

[37] J. Lai, Y. Chen, B. Han, L. Ji, Y. Shi, Z. Huang, W. Yang, and Q. Feng, "A DenseNet-based diagnosis algorithm for automated diagnosis using clinical ECG data," *Nan Fang yi ke da xue xue bao= J. Southern Med. Univ.*, vol. 39, no. 1, pp. 69–75, 2019.

[38] Z. Huang, X. Zhu, M. Ding, and X. Zhang, "Medical image classification using a light-weighted hybrid neural network based on PCANet and DenseNet," *IEEE Access*, vol. 8, pp. 24697–24712, 2020, doi: 10.1109/ACCESS.2020.2971225.

[39] R. D. Gottapu and C. H. Dagli, "DenseNet for anatomical brain segmentation," *Procedia Comput. Sci.*, vol. 140, pp. 179–185, Nov. 2018, doi: 10.1016/j.procs.2018.10.327.

[40] A. Corana. (Apr. 2016). *Architectural Evolution of NVIDIA GPUs for High-Performance Computing*. [Online]. Available: https://www.researchgate.net/publication/301363311

[41] E. Strohmaier, J. Dongarra, H. Simon, and H. Meuer. (Nov. 2018). *The Top 500 List*. [Online]. Available: https://www.top500.org/lists/2018/11

[42] J. S. Cook and N. Gupta, ''History of supercomputing and supercomputer centers,'' in *Research and Applications in Global Supercomputing*. Hershey, PA, USA: IGI Global, 2015, pp. 33–55.

[43] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, ''NVIDIA tesla: A unified graphics and computing architecture,'' *IEEE Micro*, vol. 28, no. 2, pp. 39–55, Mar. 2008.

[44] F. F. D. Santos and P. Rech, ''Analyzing the criticality of transient faults-induced SDCS on GPU applications,'' in *Proc. 8th Workshop Latest Adv. Scalable Algorithms Large-Scale Syst.*, Nov. 2017, pp. 1–7.

[45] N. DeBardeleben, S. Blanchard, L. Monroe, P. Romero, D. Grunau, C. Idler, and C. Wright, ''GPU behavior on a large HPC cluster,'' in *Proc. Eur. Conf. Parallel Process.*, 2013, pp. 680–689.

[46] L. B. Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, K. Pattabiraman, P. Rech, and M. S. Reorda, ''GPGPUs: How to combine high computational power with high reliability,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, p. 341.

[47] I. Anarado, M. A. Anam, F. Verdicchio, and Y. Andreopoulos, ''Mitigating silent data corruptions in integer matrix products: Toward reliable multimedia computing on unreliable hardware,'' *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 11, pp. 2476–2489, Nov. 2017.

[48] D. A. G. D. Oliveira, L. L. Pilla, M. Hanzich, V. Fratin, F. Fernandes, C. Lunardi, J. M. Cela, P. O. A. Navaux, L. Carro, and P. Rech, ''Radiation-induced error criticality in modern HPC parallel accelerators,'' in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 577–588.

[49] F. Kastensmidt and P. Rech, ''Radiation effects and fault tolerance techniques for FPGAs and GPUs,'' in *FPGAs and Parallel Architectures for Aerospace Applications*. Cham, Switzerland: Springer, 2016, pp. 3–17, doi: 10.1007/978-3-319-14352-1_1.

[50] *Preview Road Vehicles—Functional Safety Part 1: Vocabulary*, Standard ISO-26262, ISO 26262-1:2018, 2018. Accessed: Sep. 13, 2019. [Online]. Available: https://www.iso.org/standard/68383.html

[51] N. N. Mahatme, S. Jagannathan, T. D. Loveless, L. W. Massengill, B. L. Bhuva, S.-J. Wen, and R. Wong, ''Comparison of combinational and sequential error rates for a deep submicron process,'' *IEEE Trans. Nucl. Sci.*, vol. 58, no. 6, pp. 2719–2725, Dec. 2011.

[52] J. Noh, V. Correas, S. Lee, J. Jeon, I. Nofal, J. Cerba, H. Belhaddad, D. Alexandrescu, Y. Lee, and S. Kwon, ''Study of neutron soft error rate (SER) sensitivity: Investigation of upset mechanisms by comparative simulation of FinFET and planar MOSFET SRAMs,'' *IEEE Trans. Nucl. Sci.*, vol. 62, no. 4, pp. 1642–1649, Aug. 2015, doi: 10.1109/TNS.2015.2450997.

[53] A. P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. de Mathelin, and N. Padoy, ''EndoNet: A deep architecture for recognition tasks on laparoscopic videos,'' *IEEE Trans. Med. Imag.*, vol. 36, no. 1, pp. 86–97, Jan. 2017, doi: 10.1109/TMI.2016.2593957.

[54] A. Jin, S. Yeung, J. Jopling, J. Krause, D. Azagury, A. Milstein, and L. Fei-Fei, ''Tool detection and operative skill assessment in surgical videos using region-based convolutional neural networks,'' in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2018, pp. 691–699.

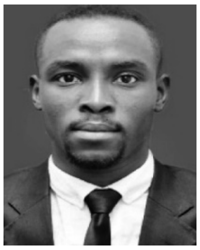
[55] S. Panigrahi, A. Nanda, and T. Swarnkar, ''A survey on transfer learning,'' in *Smart Innovation, Systems and Technologies*, vol. 194. Singapore: Springer, 2021, pp. 781–789, doi: 10.1007/978-981-15-5971-6_83.

[56] NVIDIA Corporation, ''NVIDIA GeForce GTX 980 featuring Maxwell, the most advanced GPU ever made,'' 2014, pp. 1–32. [Online]. Available: https://www.jaconnect.de/pdf/18408.pdf

[57] R. C. Baumann, ''Radiation-induced soft errors in advanced semiconductor technologies,'' *IEEE Trans. Device Mater. Rel.*, vol. 5, no. 3, pp. 305–316, Sep. 2005.

KHALID ADAM received the B.Sc. degree (Hons.) in computer engineering from the Faculty of Electrical Engineering, Alzaiem Alazhari University, Sudan, in 2013, and the M.Sc. degree from Universiti Malaysia Pahang (UMP), Malaysia, in 2016, where he is currently pursuing the Ph.D. degree with the College of Engineering. His research interests include reliability of autonomous healthcare applications, deep learning, and big data analytics.

IZZELDIN IBRAHIM MOHAMED received the Ph.D. degree in microelectronics and computer engineering from Universiti Teknologi Malaysia. He is currently a Senior Lecturer of electronics engineering with Universiti Malaysia Pahang. His research interests include networking, in particular, network traffic measurement and analysis, network management, and traffic engineering, network security, the Internet of Things (IoT) security, cloud computing, and big data security. He is also interesting in integrated circuit testing, in particular, system-level testing of embedded analogue cores in SOC as well as quantitative research method and questionnaire design and psychometrics assessment using Rasch measurement and analysis.



YOUNIS IBRAHIM received the Ph.D. degree from Hohai University, China, in 2021. He is currently working on the reliability of deep learning systems as a Free Researcher. His research interests include reliability of AI models through AI accelerators in safety-critical systems and accelerating deep learning models. More specifically, computer-vision and convolutional neural networks (CNN).