IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 14 February 2024, accepted 9 April 2024, date of publication 18 April 2024, date of current version 26 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3390564

RESEARCH ARTICLE

Hardware Design of Lightweight Binary Classification Algorithms for Small-Size Images on FPGA

SERKAN SAGLAM[1,2] AND SALIH BAYAR[2], (Member, IEEE)
[1]School of Electronics and Computer Science, University of Southampton, SO17 1BJ Southampton, U.K.
[2]Department of Electrical and Electronics Engineering, Faculty of Engineering, Marmara University, 34854 Istanbul, Turkey

Corresponding author: Salih Bayar (salih.bayar@marmara.edu.tr)

ABSTRACT This study explores the implementation of lightweight binary classification algorithms on low-cost Field-Programmable Gate Arrays (FPGAs) for medical image analysis. Recognizing the growing demand for efficient and accurate diagnostic tools in healthcare, we focus on applying FPGAs to process small-sized medical images, explicitly targeting the detection of malaria from blood cell images. Our approach involves the hardware designs of k-nearest Neighbors (k-NN), Convolutional Neural Networks (CNN), and Decision Tree classifiers rigorously tested on a publicly available Malaria dataset. The methodology emphasizes the integration of these classifiers on FPGA, detailing the optimization strategies that allow for enhanced processing speed and reduced resource utilization. Comparative analysis reveals that our FPGA-based implementation significantly outperforms MATLAB simulations, achieving processing speeds more than thousands of times faster. Our proposed hardware design also requires fewer Look-up Tables (LUTs) than other classification studies in the literature, showcasing decreases of 73.9% for k-NN, 57% for CNN, and 96.7% for the Decision Tree classifier. Furthermore, results highlight the Decision Tree classifier as the most effective, with an accuracy rate of 99.33%, followed by CNN at 97.67% and k-NN at 95.33%. These findings demonstrate the capability of FPGAs in medical image classification and underscore their potential to revolutionize disease diagnosis processes, particularly in resource-limited environments.

INDEX TERMS Binary image classification, convolutional neural network (CNN), decision tree, FPGA, hardware design, k-nearest neighborhood (k-NN).

I. INTRODUCTION

Image processing has become an essential component of various biomedical techniques used for detecting and diagnosing medical diseases [1], [2], [3]. However, studies involving image processing necessitate considerable computational power for identifying features and generating predictions [4], [5], [6], [7]. Therefore, high-performance devices such as central processing units (CPUs), graphics processing units (GPUs), and field-programmable gate arrays (FPGAs) are frequently preferred [8], [9], [10]. CPUs have inadequate parallel computing capabilities for this task [11], whereas GPUs consume a significant amount of power [12], [13]. To address this issue, FPGAs have emerged as promising alternatives for high-performance computing in image-processing applications [14], [15], [16], [17].

In digital image processing and machine learning, the development and implementation of classification algorithms have become pivotal for many applications, ranging from consumer electronics to critical medical diagnostics [18]. However, traditional algorithms' complexity and computational demands pose significant challenges, particularly in environments with limited hardware resources [19]. At this point, the concept of lightweight binary classification algorithms comes into play, addressing the need for efficient yet effective processing capabilities. These algorithms can

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

provide high accuracy while minimizing computational overhead and resource consumption, making them ideal for deployment in resource-constrained environments [20].

Using lightweight algorithms is particularly relevant in medical image analysis, where the ability to classify images accurately can have profound implications on patient diagnosis and treatment outcomes [21]. The need for rapid and reliable classification in such applications is paramount, yet often needs to be improved by the limitations of available computing resources, especially in remote or under-resourced medical facilities [22]. This research aims to bridge this gap by focusing on lightweight binary classification, offering a viable solution that balances computational efficiency with the accuracy necessary for critical medical decisions. In doing so, it endeavors to make advanced image classification more accessible and practical in scenarios where traditional, resource-intensive methods are not feasible, thereby expanding the potential for life-saving diagnostics and interventions.

FPGAs offer unique advantages, such as parallel processing capabilities and efficient use of hardware resources, which make them an attractive choice for implementing image processing algorithms for medical applications. Besides, FPGA-based image processing algorithms have demonstrated much better performance than other processing units, leading to increased interest in their use for biomedical applications [23]. In this paper, we explore using FPGAs to implement lightweight machine learning algorithms for malaria diagnosis, a widespread disease affecting millions of people globally [24].

Although new tools like Vitis, Vitis AI, and frameworks like Caffe, PyTorch, and TensorFlow exist [25], [26], [27], [28], we chose to use a pure register-transfer-level (RTL) implementation using VHDL to deliver our target designs precisely. RTL implementation on FPGA is a superior option for medical image processing to High-Level Synthesis (HLS) or software implementation on a CPU due to its area and speed performance. While HLS implementation offers a shorter design time, it is known that RTL (i.e., Hardware Description Language (HDL)) implementation provides better results in terms of both LUT usage and response time [29]. However, it is beyond the scope of this study to compare RTL with HLS implementation. Therefore, we chose RTL implementation using VHDL to apply the target designs on the Xilinx FPGA accurately. This paper highlights the importance of FPGAs as the implementation platform for machine learning algorithms, using the classification of malaria-infected blood cells as a case study. Our study contributes to a deeper understanding of massively parallel hardware platforms like FPGAs as ideal candidates for implementing machine learning algorithms.

This paper first briefly overviews the recent work in Chapter II. We described the methodology of proposed machine learning algorithms in Chapter III. Chapter IV provides information on the FPGA implementation of classifier algorithms and discusses the result of the study. Finally, Chapter V gives the conclusion.

II. RELATED WORK

FPGAs have emerged as a popular platform for accelerating machine learning studies due to their potential for high-speed computation [30]. Hence, accelerating classification algorithms on FPGA has been an essential issue. However, accelerating the classification algorithms causes an increase in the resource usage of the FPGA. Using costly FPGAs with high resources, such as LUTs, BRAM, DSP, and FF, can solve this problem. However, this solution contradicts the idea of low cost and high performance, which is the main logic of FPGA.

The common main goal of studies [31], [32], [33], [34], [35] is to optimize the CNN classification algorithm for the usage of image processing applications in embedded systems. However, it is seen that the LUT usage is excessive. The CNN classifier's classification result and classification response time change depending on the CNN model used. The structure of the convolution layer number, kernel size, number of strides, flatting layer, activation layer, etc, determines the CNN model. Increasing the convolution layers may positively affect the classification percentage as it improves learning. However, the number of Multiplication Accumulation (MAC) increases as convolutional layers increase. It causes a delay in classification time and an increase in FPGA resource usage. Especially for binary classification, reducing the number of layers to speed up classification is not the only solution. Our proposed lightweight CNN classification method designed for disease diagnosis in this article aims to improve these disadvantages.

Similarly, in k-NN studies, various methods have been proposed to increase the operating speed of the k-NN classifier [36], [37], [38], [39]. However, these studies also show high LUT usage. In one study, the authors designed a k-NN-based classification system to reduce energy consumption [39], but the system still exhibited high LUT usage. In decision tree classification studies, several methods have been proposed [40], [41], [42], [43], in which decision tree classifications are applied separately or together on FPGAs. Although the ultimate goal of these studies is to speed up the system, these studies did not propose optimized LUT usage. In particular, the study [43] aims to develop the decision tree classification method for water bodies detection. The significant problem with this approach is that it also fails to take LUTs into account.

Overall, these studies underscore the need for performance in terms of execution time in FPGA implementation. However, none of these studies have proposed an optimized classification algorithm implementation on FPGA regarding both LUT usage and execution time. Our proposed lightweight classification architectures for disease diagnosis aim to address these drawbacks. Our approach uses a simplified hardware implementation that reduces the number of convolutional layers, employs efficient activation functions, and eliminates unnecessary computations. Our method also uses FPGA's parallel processing capabilities, improving its execution time.

In summary, the reviewed literature highlights the significance of optimizing LUT usage and execution time in the FPGA implementation of classification algorithms. Our proposed method addresses this need by providing a lightweight and efficient hardware implementation for the most commonly used machine learning algorithms in image classification applications.

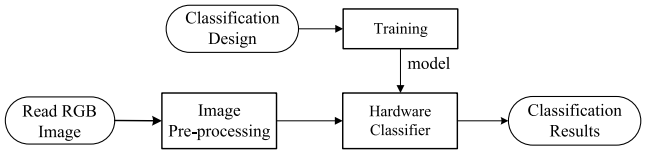


FIGURE 1. Flowchart of hardware-based classifier system.

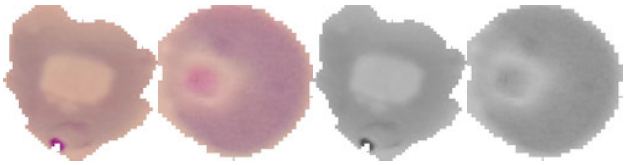
III. METHODOLOGY

This section describes the methodology of classification methods used in this study; we indicate the general classification steps of medical images in Figure 1. We first enhance the RGB image according to the classification method in the pre-processing stage. Then, the enhanced image is fed to the hardware classifier implemented on the FPGA. One of the other inputs of the hardware classifier is the training model we already implemented at design time outside the FPGA. The proposed lightweight hardware classifiers decide the class according to the enhanced image pixel values and the trained model parameters on the FPGA. We further explain the hardware designs in depth in the following paragraphs, one by one.

A. BINARIZATION METHOD

The image sizes used in the dataset are very diverse, ranging from 76 to 214. To make it easier to process, we chose the images' dimensions as 8×8, 16×16, 32×32, 64×64, and 128×128 for each experiment before converting them to grayscale images.

The RGB color space is the most popular of the color spaces, and in this color space, each pixel takes 256 different values between 0 and 255 of 8 bits. Further, the value 0 refers to the black color and the value 255 to the white color. Since it requires more computation to process an RGB image directly as it contains a lot of information, it would be more time-efficient to convert RGB values to grayscale in the pre-processing step. The grayscale image's basic logic is that the red, green, and blue intensity in RGB is equal. Thus, it has a single intensity value, unlike the three intensities required to indicate a pixel. Hence, before processing, we converted diseased and healthy RGB images to grayscale as in Figure 2.



(a) Diseased RGB (b) Healthy RGB (c) Diseased GS (d) Healthy GS

FIGURE 2. Diseased and healthy RGB/ Gray Scale (GS) images.

We observed that grayscale image processing gave inadequate accuracy rates in our initial experiments. Therefore, we preferred to convert the grayscale image to a binary format to get the region of interest in the image more precisely. The binarization method converts a grayscale image between 0 and 255 into a binary image containing two-pixel classes only in 0 (black) and 1 (white). This method provides more accuracy because it reduces the noise significantly in the original image [44] and has a critical role in image processing [45].

The same binarization methods provide different performances in different datasets. Generally accepted methods for binarization methods include the Fixed Thresholding Method, Herbaceous Method, Kittler Method, Niblack Method, Adaptive Method, Sauvola Method, and Bernsen Method [46]. The binarization method that offers the best results can often be associated with the data set. Therefore, one of the most challenging parts of this work was deciding which binarization method was the most suitable for our data set. We preferred to use the fixed threshold due to its efficiency and simplicity in this work. In the binarization method, one can determine the $B(x, y)$ value by a predefined threshold value, T. The methodology followed for the selection of T changes the produced binary image. Hence, the T value is an important criterion when converting a grayscale image to a binary image [46].

$$B(x, y) = \begin{cases} 1 & f(x, y) \geq T \\ 0 & f(x, y) < T \end{cases} \quad (1)$$

As it is apparent from Eq. (1), we compare each pixel value of the grayscale image with T. Here, we select the T value according to the binary coefficient. If the grayscale image's pixel value is greater than or equal to T, the result is one; otherwise, it is zero. Hence, one of the most critical steps in this process is ascertaining the optimum T. We observed that there is no fixed optimum T for all image sizes and algorithms. Hence, we found the optimum T for each image size and machine learning algorithm by brute force searching in the training phase. For example, we concluded that the optimum T=140 for 16×16 image size in k-NN or T=127 for 64×64 image size in the CNN algorithm.

B. K-NN

Figure 3 illustrates the proposed architectures of both k-NN and CNN classifiers. As seen in *Step* 1 of Figure 3, the pre-processing part is the same for both classifiers. For k-NN and CNN, we converted the RGB image to a grayscale image in *Step* 1*a*. Then, we transformed the image into a binary image by applying the binarization method in *Step* 1*b*. In the last stage of the pre-processing step (*Step* 1*c*), we unified the image sizes as the input images have different sizes. Note that we implemented *Step* 1 only in MATLAB, not in the hardware. Hence, we did not include its processing time in our calculations in the experiments.

Although k-NN and CNN architectures have common pre-processing parts, other processes differ for both

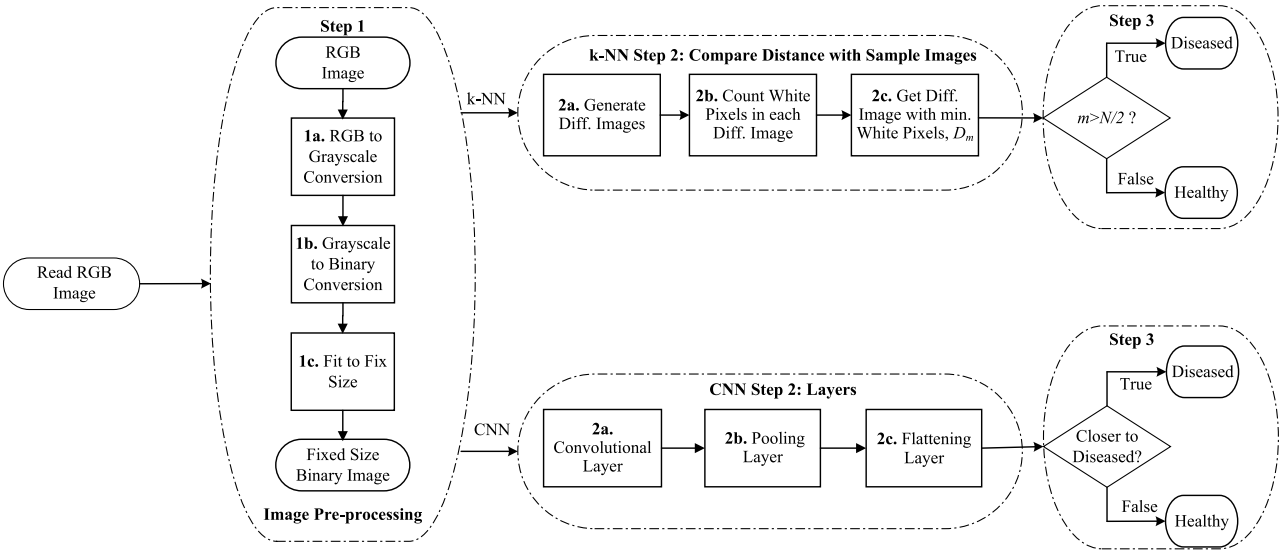


FIGURE 3. Block diagram of proposed k-NN and CNN classifiers.

classifiers. We first propose the k-NN architecture in this section. The architecture of the CNN classifier will be investigated in detail in Section III-C.

The k-NN algorithm is a controlled machine-learning method that classifies data according to the training set. One can determine required parameters with pre-training procedures in most supervised classification algorithms. Therefore, in the k-NN algorithm, a training set is first created with the help of the labeled data. Then, the parameter of k and a *distance function* is selected. It is well-known that choosing an optimum k-value is not easy; a low value of k may lead to a more significant influence on the result, whereas a considerable value of k makes the design computationally intensive. We observed that k=1 gives the best accuracy in our binary classification problem. Besides, setting k as 1 in k-NN requires the lowest computational load regarding both area and speed. Hence, we selected k as 1 in k-NN's MATLAB and hardware implementations and named our proposed k-NN model lightweight.

Since we converted input images to binary images in the pre-processing step as presented in *Step* 1 of Figure 3, applying distance functions such as Minkowski, Euclidean, and Manhattan would give the same result. Hence, we did not involve any distance function to compare binary images; we used these functions only for grey-scale images in the initial attempts while developing the algorithm. We described the outcomes obtained from the analytical procedures of distance methods in Section IV.

We compared the test image with the labeled images stored in the ROM in k-NN's *Step* 2 of Figure 3. The first half of the tagged images are healthy images at the locations starting from ROM_1 to $ROM_{N/2}$. In contrast, the second half of the labeled images are diseased images at the locations starting from $ROM_{N/2+1}$ to ROM_N. Firstly, we generate the difference of the test image with each labeled image in k-NN's *Step* 2*a*. Then, we counted white pixels in each difference image (k-NN's *Step* 2*b*). As the last substep of *Step* 2, we extracted the difference image with a minimum number of white pixels, D_m (k-NN's *Step* 2*c*). In the last step of the k-NN algorithm (k-NN's *Step* 3), we compared the index value m with the ROM index value of $N/2$. If $m > N/2$, then the difference image with a minimum number of white pixels is closest to the diseased images, indicating that the image under test is also diseased. Otherwise, it is a candidate for a healthy image.

Figure 4 presents the minified hardware of the k-NN architecture. Here, while the X_{nxn} matrix represents the image under test, the blocks named ROM_1 to ROM_N provide labeled images stored in the ROM. After RGB to binary conversion and image scaling operations in pre-processing (see *Step* 1 in Figure 3), we calculated the differences between labeled image values with the size *nxn* and the X_{nxn} matrix (see *Step* 2*a* in Figure 4). To count the number of white pixels, we take the cumulative sum of the absolute difference in pixels between the two images (*Step* 2*b* in Figure 4) as given in Eq. 2.

$$D_i = \sum_{j=0}^{nxn} \left| ROM_{i,j} - X_j \right| \tag{2}$$

We expressed the difference between i^{th} labeled image ROM_i and test image X_{nxn} with a single non-negative integer, D_i. As N- labeled images exist in the ROM, we have N comparison results as integer values. We pick the minimum of these values as shown in Eq. 3 and related memory index m (*Step* 2*c* in Figure 4).

$$min(\sum_{j=0}^{nxn} \left| ROM_{1,j} - X_j \right|, \ldots, \sum_{j=0}^{nxn} \left| ROM_{N,j} - X_j \right|) \tag{3}$$

If this index is greater than the $N/2$, then we conclude that the tested image belongs to the image class of diseased set; otherwise, we come up with the result that it is a healthy image (*Step* 3 in Figure 4).

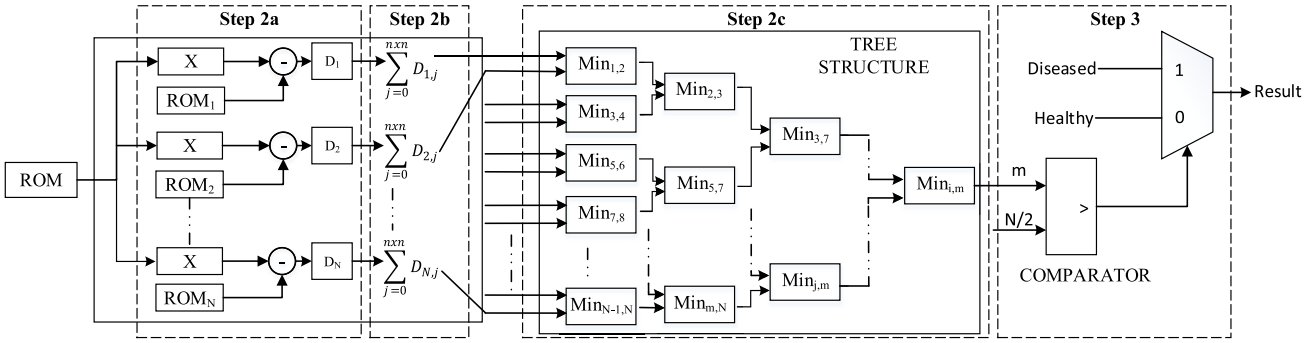


FIGURE 4. Proposed hardware of minified k-NN architecture.

C. CNN

We can broadly define CNN as a deep neural network that classifies various objects in images and videos. Besides, it is the most preferred classifier in object recognition in image processing applications [47]. The bottom half of Figure 3 represents the process of our proposed lightweight CNN classification method. After getting the input image and applying the pre-processing step (*Step* 1), the image will be processed further in the following layers: convolution layer (CNN's *Step* 2*a*), pooling layer (CNN's *Step* 2*b*), and flattening layer (CNN's *Step* 2*c*).

The convolution layer (CNN's *Step* 2*a* in Figure 3) eliminates unnecessary level features in the image by applying the filter selected for disease detection. It is well-known that 2D digital filters are primarily square and have odd side lengths to keep the equal number of surrounding neighbour pixels for the interested pixel; the most lightweight and efficient filter can be in size 3×3. Hence, we determined this filter as 3×3 to detect malaria-infected blood cells. We did not use any well-known filters such as Gaussian or Median Filter; instead, we trained the CNN using labelled images and optimized the weights of the filter matrix. We placed the matrix values of the filter in the upper left part of the binary image matrix, and the same index values are multiplied by each other. For example, we multiplied the first filter index by the first index of the binary image. We took the sum of the values formed by the product of the indexes and stored it as the first index of the feature map. We repeated this process so that the filter matrix is shifted to the right one at a time. After finishing the binary image row, we scrolled down an index to start from the far left, and we repeated the same process until the feature map was completed [48].

The matrix formed in the convolution layer is called a feature map and needs padding before the pooling layer. The padding method (CNN's *Step* 2*b* in Figure 3) is required to preserve the information in the convolution matrix formed in the first layer of CNN before the pooling layer. Also, this method should be applied to ensure that the feature map matrix size and the input image's matrix size are the same.

The pooling layer (CNN's *Step* 2*b* in Figure 3) allows controlling incompatibility in the network by minimizing the network's number of computations and parameters. We preferred to use the Max-Pooling technique in this work because it is the most popular pooling method and gives better results than other pooling methods [49]. The fundamental working essential of the Max-Pooling technique is to divide the matrix obtained by the padding method into matrices of specified same sizes and create a new matrix by selecting the highest value in these matrices.

In a regular CNN implementation, the flattening layer (CNN's *Step* 2*c* in Figure 3) generates input data for the fully connected layer for which the classification result is determined. The 2D matrix obtained by applying the Max-Pooling method is transformed into a Single Column Vector (SCV) in this layer. Implementation details of these three layers are explained in detail in our previous study [50].

According to the experimental results, we observed that healthy images produce a very similar SCV as the output of the flattening layer. Hence, we took the average SCVs of labelled healthy images in our CNN implementation training phase and named them H-SCV. Similar to this, we applied the same operation for labelled diseased images and called it D-SCV. Instead of using a fully connected layer as in a standard CNN implementation, we preferred to compare SCV generated by test image with H-SCV and D-SCV (CNN's *Step* 3 in Figure 3). As our proposed CNN architecture lacks a fully connected layer, its design was simple and occupied very little hardware area. Hence, we named our proposed CNN model as lightweight.

Figure 5 demonstrates the minified hardware of the proposed CNN classification. First, we processed the input image with the filter matrix and formed the convolution matrix (*Step* 2*a* in Figure 5). Since the matrix index formed in the convolution layer is decreased, the indexes are increased in the padding layer (*Step* 2*b* in Figure 5). Moreover, in the max-pooling part (*Step* 2*b* in Figure 5), the highest values are obtained by dividing them into 2×2 matrices, and the following step in the flattening part (*Step* 2*c* in Figure 5) is performed to get SCV representation of the test image. If the difference between the test image's SCV and H-SCV is less than the difference between the test image's SCV and D-SCV, we concluded that the test image is healthy. Otherwise, we decided it to be a diseased image (*Step* 3 in Figure 5).

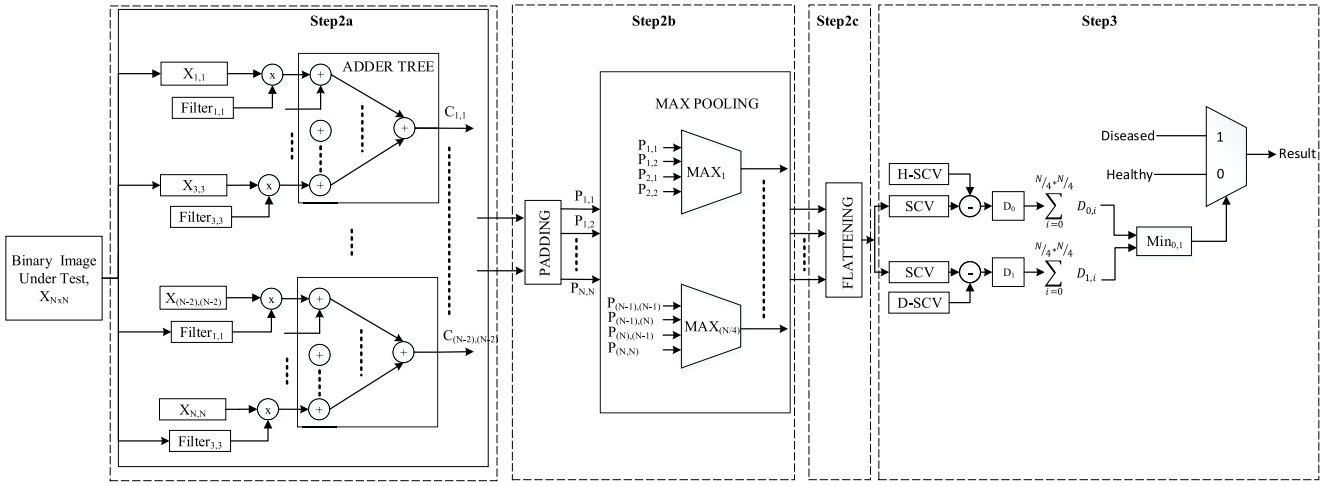


FIGURE 5. Proposed hardware of minified CNN architecture.

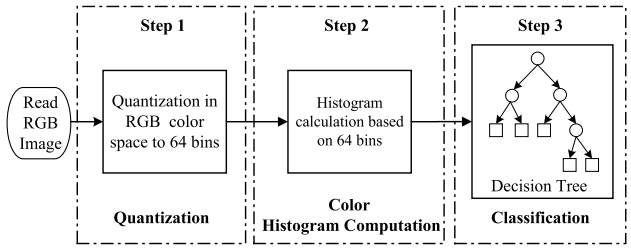


FIGURE 6. Proposed architecture of decision tree classifier.

D. DECISION TREE

In a decision tree, while the nodes represent the distinguishing features or attributes of the classification system, the leaves are the final decision results [51]. A decision tree structure is created using the training data and attribute information at design time. The training phase, in which we created the decision tree, will be detailed further in subsection III-D1.

After a successful training phase, to determine which class the test data belongs to, we applied test image to be observed as an input to the decision tree created in the training phase. We tested the data from the starting root node to the sub-nodes according to the decisions given by intermediate nodes. This process continued until we reached a specific leaf of the tree, and the classification is concluded. The implementation details of the test phase will be given in subsection III-D2.

1) TRAINING PHASE OF DECISION TREE

To extract meaningful data from the training images, we quantized each training image and calculated their histograms at design time. After the colour histogram calculation, we observed that all healthy images generate the same histogram values for the histogram bins: Bin-1, Bin-17, Bin-22, Bin-38, Bin-39, Bin-42, Bin-43, and Bin-59. However, this is not the same case for the diseased images; for example, 16% of Bin-39 and 30% of Bin-17 has the same value for both diseased and healthy images. According to our experiments, we observed that Bin-42 and Bin-59 have the least common histogram values in diseased and healthy images. As a result, we generated the decision tree using these two bins having a 99.33% accuracy.

2) TEST PHASE OF DECISION TREE

Figure 6 presents the steps taken in the test phase of the decision tree classification for the observed image. After getting the test RGB image as an input to the system, the essential step is the feature extraction from the image. Colour histograms are the most common usual image descriptors [52] for the feature extraction process. However, as the test image may have millions of pixels (e.g. an image having 1920 × 1080 pixels), attempting to extract data from each pixel directly would be an inefficient way. Hence, we quantized the test image (*Step* 1) before applying the colour histogram to it (*Step* 2). Upon the colour histogram calculation step, we applied the extracted colour histogram values of the image under the test to the decision tree (*Step* 3) whose nodes are decision points according to the various histogram values.

From the Figure 7 above, we can see the hardware implementation of decision tree-based classification in detail. We preferred to use 64-color RGB, i.e. 6-bit RGB image, to limit the number of dimensions of the image under test by applying the quantization process. So, we reduced the number of bins in the colour histogram from 256 to 64 for an 8-bit RGB test image (*Step* 1).

After getting 6-bit quantized image pixels, we calculate each pixel's colour bin one by one successively in the histogram computation step (*Step* 2). To select each pixel's quantized value one by one sequentially, we used a 6-bit NxN to $log_2(NxN)$ multiplexer. The output of a $log_2(NxN)$ - bit counter is connected to the select bit of this multiplexer so that we were able to feed the colour histogram circuit for each quantized pixel value one by one successively. Instead of having such a multiplexer, we could also store the 6-bit quantized pixel values in the memory. In the colour

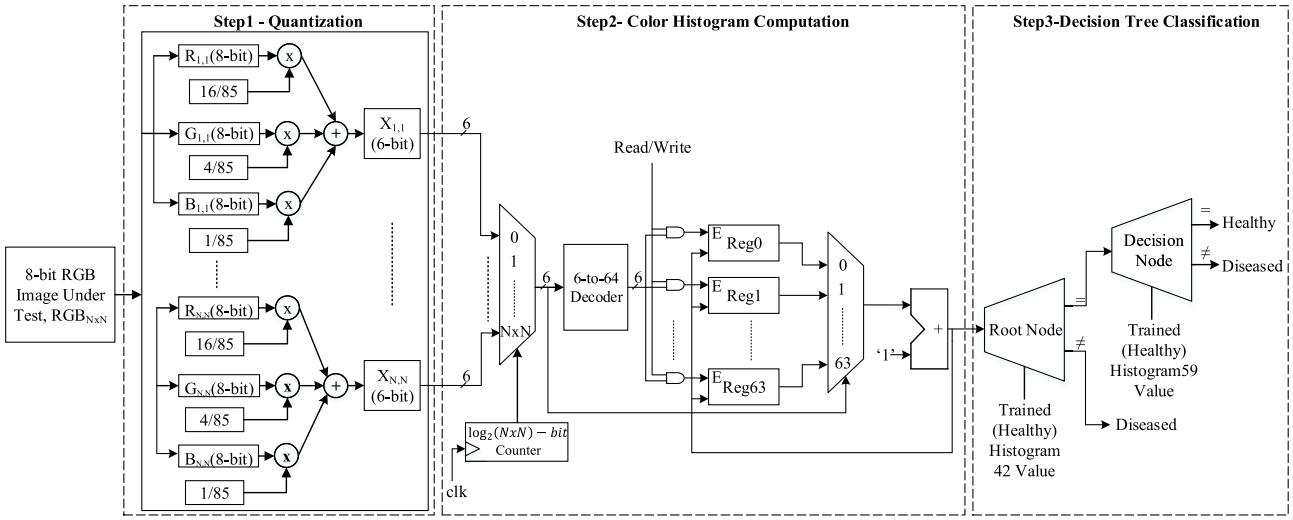


FIGURE 7. Proposed hardware of decision tree architecture.

histogram circuit, we used the idea of a register file by reading and writing the corresponding bin number as a register for each pixel (*Step* 2). As evident from the colour histogram calculation perspective, the value of each pixel determines the distribution in the corresponding colour. So, we assigned each pixel of the image to the specified colour bin by increasing the value of the corresponding bin.

In the final step of the decision tree based binary classification, we applied corresponding histogram values of the test image to the decision tree generated in the design time (*Step* 3). If the test image's colour histogram value at 42 is not equal to the one generated by the trained data, then we concluded that the test image is diseased. Otherwise, we made a successive comparison, in which we consider the equality of the test image's colour histogram value at 59 with the one generated by the trained data. If they matched, we decided that the test image was healthy; otherwise, we concluded that it was a diseased image.

IV. RESULTS AND DISCUSSION

For each classification method, we used 300 images; half of these images were healthy, and the remaining half were diseased. We selected 80% of this data set as the training set and chose the remaining 20% of these images as the test set. We performed the training operation in MATLAB on a Windows computer with the following specs: Intel Core i5-3337U @ 2.7GHZ with 8GB DDR3 RAM. We implemented the hardware of the classification methods in the Vivado 2021.2 using the Xilinx Zynq-7000 SoC ZC702 FPGA in simulations.

This work tested k-NN, CNN, and decision tree classification methods on 8 × 8, 16 × 16, 32 × 32, 64 × 64, and 128 × 128 pixel images to determine malaria. Besides, we observed the effect of the binarization threshold value on k-NN and CNN classification results.

At first, we tested grayscale images for different distance functions for the k-NN algorithm; Manhattan had an accuracy percentage of 50.33%, Euclidean 52.66%, and Minkowski 57.33%. Although the Minkowski equation had the highest value among these three equations, the accuracy rate 57.33% was insufficient for malaria disease detection.

As with the k-NN classification, we tested the grayscale image in the CNN classification, and the highest percentage of accuracy was 52.34%. Therefore, we concluded that the binarization method was needed to predict k-NN and CNN algorithms better. Applying the binarization method improves the classification quality with the k-NN algorithm. Since the examined image is binary, the absolute difference between the two pixels is zero or one. Thus, the exponential value of 0-1 numbers will not change for any value of D_i; the result is the same for Manhattan, Euclidean, and Minkowski distance equations. Hence, we did not involve any distance function in comparing binary images.

Table 1a and 1b show each implementation's accuracy rate and LUT utilization, respectively. Here, we propose the optimum results considering the accuracy rate and design area. To highlight these optimum results, we present them in boldface type in Table 1a and 1b. Note that ∗ symbols in Table 1b indicate infeasible FPGA implementations due to very long synthesis time or excessive LUT usage. As seen in Table 1a, the k-NN classification method achieves the highest accuracy rate of 95.33% at 64 × 64 and 128 × 128 image sizes. However, applying k-NN on these images is impossible due to infeasible LUT usage; the related designs do not fit in the Xilinx Zynq-7000 SoC ZC702 FPGA (see Table 1b). Besides, the best practicable design for k-NN is 94.67% at 32 × 32 image size. Nevertheless, the k-NN implementation with 32 × 32 image size occupies 39.26% of the whole FPGA (see Table 1b). Hence, we can conclude that the optimum result in terms of both accuracy rate and hardware area for k-NN can be the design at 16 × 16 image size having 92.00%

TABLE 1. Detection accuracy rate and utilization of LUTs of classification methods on binary images.

Image Size	Accuracy Rate (%)		
	k-NN	CNN	DT
8x8	68.00	74.33	**99.33**
16x16	**92.00**	**96.33**	99.33
32x32	94.67	96.67	99.33
64x64	95.33	97.33	99.33
128x128	95.33	97.67	99.33

(a) Detection accuracy rate of k-NN, CNN, and decision tree (DT) classification methods on binary images

Image Size	Utilization of LUTs (%)		
	k-NN	CNN	DT
8x8	1.96	2.79	**2.12**
16x16	**9.33**	**11.65**	18.78
32x32	39.26	46.32	*
64x64	*	*	*
128x128	*	*	*

(b) Utilization of LUTs of k-NN, CNN, and decision tree (DT) RTL implementations with binary images on the Xilinx Zynq-7000 SoC ZC702 FPGA

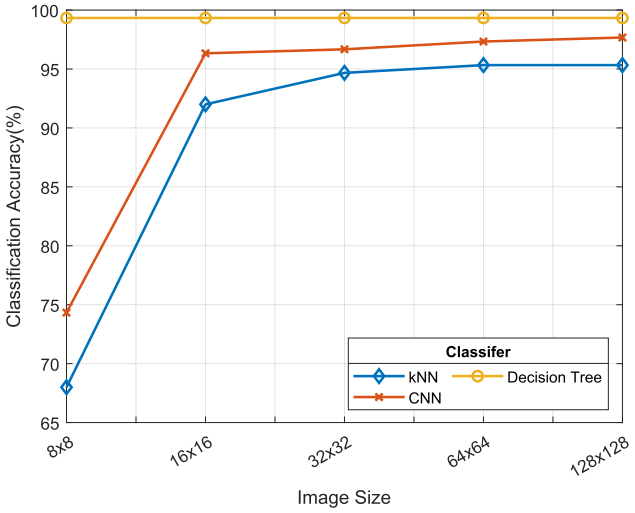


FIGURE 8. Classification accuracy rate based on image size.

accuracy (see Table 1a) with the area occupation of solely 9.33% (see Table 1b).

The CNN classification has the highest detection percentage with 97.67% with a 128 × 128 image size. However, 16 × 16 pixel designs are more realistic than 128 × 128 pixel designs as the hardware implementation of CNN for an image size greater than 32 × 32 does not fit into the target hardware. Unlike k-NN and CNN classifiers, we obtained the same best result in the decision tree for each different image size. As seen in Table 1a, the decision tree classifies the images in any size at the accuracy rate of 99.33%. Since all designs gave the same result, we chose the design with the 8 × 8 image size having the area occupation of solely 2.12% of the target FPGA.

One should choose the pixel dimensions of these classifiers according to the priority of the design. For example, while the k-NN classifier accuracy rate at 32 × 32 pixels is 94.67%, this rate is 92% at 16 × 16 pixels. However, when examined using LUTs in Table 1b, it is a severe saving of 320% compared to 16 × 16 pixels and 32 × 32 pixels. According to the obtained result, we can conclude that the most suitable pixel sizes on FPGA designs are 16 × 16 for k-NN, 16 × 16 for CNN, and 8 × 8 for the decision tree.

Figure 8 displays the accuracy rates of each proposed method concerning various image sizes. Here, we can see that the decision tree classification method has the highest accuracy rate, with 99.33%. The CNN follows it with 97.67% and the k-NN with 95.33%. Note that the accuracy rates are the same for MATLAB and hardware implementations. As we could not get the results of infeasible implementations (indicated with ∗ symbols in Table 1b) on FPGA, we used MATLAB accuracy results directly. Although finding the best classification results pixel values is a critical step, one should interpret these results cautiously when designing hardware because the architecture of designs with 64 × 64 and 128 × 128 pixels takes up an extensive amount of occupied areas of the FPGA. Since 53200 LUTs of the Xilinx Zynq-7000 SoC ZC702 used in this study are available, designs for 64 × 64 and 128 × 128 pixel values are unrealistic. Hence, we examined designs of 8 × 8, 16 × 16, and 32 × 32 pixels for k-NN and CNN. The decision tree's hardware exceeds the LUT limit even for 32 × 32 pixels. Therefore, we only tested 8 × 8 pixels and 16 × 16 pixels of hardware for the decision tree classifier.

While Figure 9a shows the execution times of each classification algorithm, Figure 9b indicates the LUT usage on FPGA for the designs with image sizes 8 × 8, 16 × 16 and 32 × 32. As can be seen, 8 × 8 pixel designs use the least LUTs and are the fastest. However, this result does not mean the 8 × 8 pixel design is ideal for all classification methods. Only the decision tree classification method reached the highest accuracy percentage at 8 × 8 pixels. However, the perfect design for k-NN and CNN is 16 × 16 pixels.

Figure 9c demonstrates the classification algorithms' running times of a single image on MATLAB and FPGA. As it is evident from this figure, the fastest hardware design is k-NN with 44.72 *ns*, CNN 54.378 *ns*, and decision tree 336.62 *ns*. The classifier's time to detect an image in MATLAB is 69103 *ns* for k-NN, 275103 *ns* for CNN, and 591103 *ns* for the decision tree. The most prominent finding from the analysis is that the classification algorithms implemented on FPGA are much faster than the ones on MATLAB. When we interpret the experimental results carefully, the outcomes procured on FPGA are more rapid than those obtained by MATLAB up to 5059 times for CNN (=275103/54). The percentage of accuracy rate is a high priority for a system that aims to reduce the number of people who die from malaria. However, evaluating only the percentage of accuracy is not an ideal approach for this purpose. Besides, metrics such as speed and occupied areas of the hardware play an essential role in the system's quality.

Comparisons of the implementations of k-NN, CNN, and decision tree classifiers on FPGA are demonstrated in

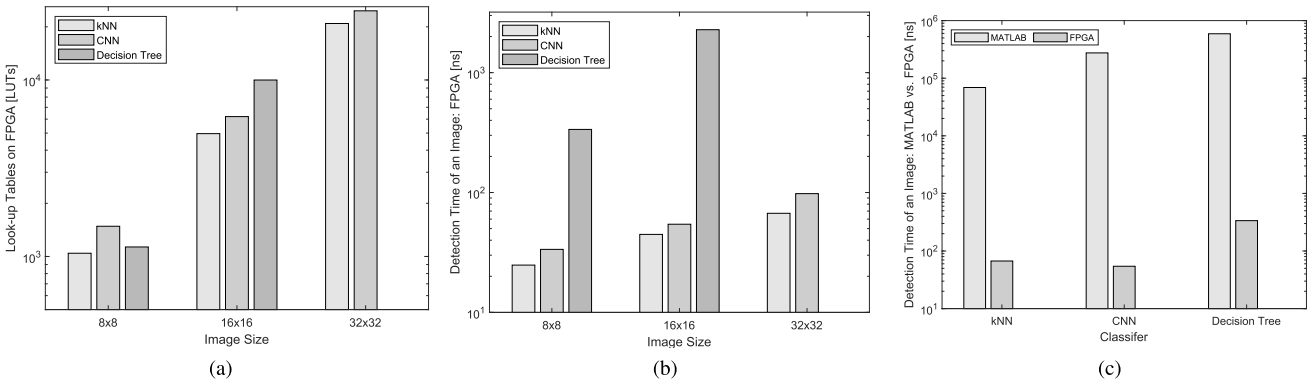


FIGURE 9. (a) Classification detection time of an image on FPGA based on image size (b) Resource utilization of proposed implementations on FPGA [LUTs] (c) Classification detection time of an image: FPGA vs. MATLAB.

TABLE 2. Comparison with k-NN implementations on FPGA.

Study	[36]	[37]	[39]	Our Work
Precision	N/A	N/A	N/A	8-bit fixed
Frequency	N/A	N/A	N/A	165MHz
FPGA Chip	XC7Z020	XC7Z020	Zynq-7000	XC7Z20
DSP Util.	N/A	N/A	7	0
Logic Util.	36K	35K	19K	**5K**
On-chip BRAM	48	36	39	0

TABLE 3. Comparison with CNN implementations on FPGA.

Study	[31]	[32]	[33]	[34]	[35]	Our Work
Precision	32bit float	8-16-bit fixed	16-bit fixed	Q15	8-bit fixed	8-bit fixed
Frequency	100MHz	100MHz	156MHz	150MHz	160MHz	165MHz
FPGA Chip	VX485T	Stratix V GXA7	VX690T	XC7Z045	XC7Z20	XC7Z20
DSP Util.	2240	256	2144	391	134	0
Logic Util.	186K	121K	273K	77K	34K	**24K**
On-chip BRAM	1086	1552	956.5	545	133.5	0

TABLE 4. Comparison with decision tree implementations on FPGA.

Study	[40]	[41]	[42]	Our Work
Precision	N/A	32bit float	N/A	8-bit fixed
Frequency	357 MHz	142.86MHz	N/A	165MHz
FPGA Chip	Zynq-7000 SoC	Zynq-7000 SoC	Zynq-7000 SoC	XC7Z20
DSP Util.	256	48	8	0
Logic Util.	81K	35K	11K	**9K**
On-chip BRAM	N/A	75	18	0

Table 2, Table 3, and Table 4. In this study, we designed hardware to detect the disease without using BRAM and DSP. Our k-NN hardware design uses 7.33x [36], 7.24x [37], and 3.98x [39] fewer LUTs than the studies in the Table 2, respectively. When CNN FPGA implementations are examined in detail in Table 3, it is seen that our study reached the highest operating frequency. Furthermore, our CNN hardware implementation has a significant advantage in using LUTs efficiently. Our CNN hardware uses 7.56x [31], 4.91x [32], 11.09x [33], 3.13x [34], 1.38x [35] fewer LUTs, respectively than the studies in Table 3. When Table 4 is examined in detail, our decision tree hardware design's success in using LUTs is clearly seen. Our decision tree hardware has usage of LUTs 8.1x [40], 3.58x [41], and 1.18x [42] less respectively.

These results demonstrate our main goal for efficiently using LUTs for hardware designs of k-NN, CNN, and decision trees. Although compared works use different data sets for diverse problems, we demonstrated that our proposed circuits occupy tiny chip areas for standard classifiers. Hence, our proposed designs can be efficiently run on even small, low-cost FPGAs.

V. CONCLUSION

The primary purpose of this study was to perform binary image classification on low-cost FPGA devices by classifying small-size images with lightweight machine learning algorithms: simplified k-NN, CNN, and decision trees. We executed all necessary training steps on the desktop computer using MATLAB and implemented tests on both computer and FPGA. As a result of these tests, we observed that the most successful classifier is the decision tree, with an accuracy rate of 99.33% of detecting diseased images. CNN follows it with 97.67%, and k-NN with 95.33%. We found that the decision tree is the slowest classifier with 336.62ns while implementing various classification algorithms on FPGA. In Comparison, the k-NN has the fastest classification time with 44.72 ns. Also, we investigated the classification methods on MATLAB and observed that the algorithms implemented on the FPGA device are 5059 times faster.

The presented work shows that FPGAs can be considered powerful machine-learning processing platforms. However, it should be attentive to the circuit size built on the FPGA. The supreme advantage of this research is that fewer LUTs are used compared to other studies. We achieved this by keeping the design simple in all hardware implementations. Consequently, this benefit provides medical image processing on FPGA devices with fewer LUTs. Mainly, when compared to other studies, it has been observed that the proposed system offers savings in the number of LUTs used by the classifiers applied on the FPGA devices. Classifiers' LUT savings are 73.9% for k-NN, 57% for CNN, and 96.7% for the decision tree.

The evidence from this study suggests that when hardware designers develop accurate hardware designs for classifiers, performing lightweight machine learning algorithms for

medical image processing on a low-cost FPGA is feasible with satisfied speed and LUT area usage.

Although the current study is based on a small sample of diseased malaria cells, the findings suggest that machine learning algorithms for image processing are very efficient and feasible on the FPGAs when a proper design is selected.

REFERENCES

[1] S. R. Nayak, D. R. Nayak, U. Sinha, V. Arora, and R. B. Pachori, "Application of deep learning techniques for detection of COVID-19 cases using chest X-ray images: A comprehensive study," *Biomed. Signal Process. Control*, vol. 64, Feb. 2021, Art. no. 102365.

[2] P. Samant and R. Agarwal, "Machine learning techniques for medical diagnosis of diabetes using iris images," *Comput. Methods Programs Biomed.*, vol. 157, pp. 121–128, Apr. 2018.

[3] M. Poostchi, K. Silamut, R. J. Maude, S. Jaeger, and G. Thoma, "Image analysis and machine learning for detecting malaria," *Transl. Res.*, vol. 194, pp. 36–55, Apr. 2018.

[4] S. Wang, D. M. Yang, R. Rong, X. Zhan, J. Fujimoto, H. Liu, J. Minna, I. I. Wistuba, Y. Xie, and G. Xiao, "Artificial intelligence in lung cancer pathology image analysis," *Cancers*, vol. 11, no. 11, p. 1673, Oct. 2019.

[5] J. H. Thrall, X. Li, Q. Li, C. Cruz, S. Do, K. Dreyer, and J. Brink, "Artificial intelligence and machine learning in radiology: Opportunities, challenges, pitfalls, and criteria for success," *J. Amer. College Radiol.*, vol. 15, no. 3, pp. 504–508, Mar. 2018.

[6] M. Avanzo, L. Wei, J. Stancanello, M. Vallières, A. Rao, O. Morin, S. A. Mattonen, and I. El Naqa, "Machine and deep learning methods for radiomics," *Med. Phys.*, vol. 47, no. 5, pp. 185–202, May 2020.

[7] L. N. Sanchez-Pinto, Y. Luo, and M. M. Churpek, "Big data and data science in critical care," *Chest*, vol. 154, no. 5, pp. 1239–1248, Nov. 2018.

[8] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, pp. 1–74, Mar. 2021.

[9] A. J. A. AlBdairi, Z. Xiao, A. Alkhayyat, A. J. Humaidi, M. A. Fadhel, B. H. Taher, L. Alzubaidi, J. Santamaría, and O. Al-Shamma, "Face recognition based on deep learning and FPGA for ethnicity identification," *Appl. Sci.*, vol. 12, no. 5, p. 2605, Mar. 2022.

[10] F. Behbahani, A. Behrad, and M. H. Moaiyeri, "An ultra-fast and energy-efficient CNTFET-based image corner detection hardware for real-time image processing applications," *AEU Int. J. Electron. Commun.*, vol. 175, Feb. 2024, Art. no. 155099. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1434841123005757

[11] L. Chen, X. Wei, W. Liu, H. Chen, and L. Chen, "Hardware implementation of convolutional neural network-based remote sensing image classification method," in *Proc. 7th Commun., Signal Process., Syst.* Dalian, China: Springer, 2018, pp. 140–148.

[12] L. Li, S. Zhang, and J. Wu, "Efficient object detection framework and hardware architecture for remote sensing images," *Remote Sens.*, vol. 11, no. 20, p. 2376, Oct. 2019.

[13] N. Zhang, X. Wei, H. Chen, and W. Liu, "FPGA implementation for CNN-based optical remote sensing object detection," *Electronics*, vol. 10, no. 3, p. 282, Jan. 2021.

[14] Q. Liu, J. Setter, D. Huff, M. Strange, K. Feng, M. Horowitz, P. Raina, and F. Kjolstad, "Unified buffer: Compiling image processing and machine learning applications to push-memory accelerators," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 2, pp. 1–26, Jun. 2023.

[15] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, "A systematic literature review on hardware implementation of artificial intelligence algorithms," *J. Supercomput.*, vol. 77, no. 2, pp. 1897–1938, Feb. 2021.

[16] D. Ali, A. U. Rehman, and F. H. Khan, "Hardware accelerators and accelerators for machine learning," in *Proc. Int. Conf. IT Ind. Technol. (ICIT)*, Oct. 2022, pp. 01–07.

[17] H. Yun and D. Park, "Low-power lane detection unit with sliding-based parallel segment detection accelerator for FPGA," *IEEE Access*, vol. 12, pp. 4339–4353, 2024.

[18] S. Karakanis and G. Leontidis, "Lightweight deep learning models for detecting COVID-19 from chest X-ray images," *Comput. Biol. Med.*, vol. 130, Mar. 2021, Art. no. 104181.

[19] D. Preuveneers, I. Tsingenopoulos, and W. Joosen, "Resource usage and performance trade-offs for machine learning models in smart environments," *Sensors*, vol. 20, no. 4, p. 1176, Feb. 2020.

[20] D. Yu, Q. Xu, H. Guo, C. Zhao, Y. Lin, and D. Li, "An efficient and lightweight convolutional neural network for remote sensing image scene classification," *Sensors*, vol. 20, no. 7, p. 1999, Apr. 2020.

[21] S. Bhattacharjee, C.-H. Kim, D. Prakash, H.-G. Park, N.-H. Cho, and H.-K. Choi, "An efficient lightweight CNN and ensemble machine learning classification of prostate tissue using multilevel feature analysis," *Appl. Sci.*, vol. 10, no. 22, p. 8013, Nov. 2020.

[22] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "MedMNIST v2—A large-scale lightweight benchmark for 2D and 3D biomedical image classification," *Sci. Data*, vol. 10, no. 1, p. 41, Jan. 2023.

[23] A. Sanaullah, C. Yang, Y. Alexeev, K. Yoshii, and M. C. Herbordt, "Real-time data analysis for medical diagnosis using FPGA-accelerated neural networks," *BMC Bioinf.*, vol. 19, no. 18, pp. 19–31, Dec. 2018.

[24] J. Talapko, I. Škrlec, T. Alebić, M. Jukić, and A. Včev, "Malaria: The past and the present," *Microorganisms*, vol. 7, no. 6, p. 179, Jun. 2019.

[25] M. Ghiglione, V. Serra, A. Raoofy, G. Dax, C. Trinitis, M. Werner, M. Schulz, and G. Furano, "Survey of frameworks for inference of neural networks in space data systems," in *Proc. Data Syst. Aerosp. (DASIA). Eurospace*, 2022. [Online]. Available: https://www.bgd.ed.tum.de/pdf/2022_cnnInSpace_Ghiglione.pdf

[26] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement.*, vol. 16, Savannah, GA, USA, 2016, pp. 265–283.

[27] M. Verucchi, G. Brilli, D. Sapienza, M. Verasani, M. Arena, F. Gatti, A. Capotondi, R. Cavicchioli, M. Bertogna, and M. Solieri, "A systematic assessment of embedded neural networks for object detection," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2020, pp. 937–944.

[28] J. Wang and S. Gu, "FPGA implementation of object detection accelerator based on vitis-AI," in *Proc. 11th Int. Conf. Inf. Sci. Technol. (ICIST)*, May 2021, pp. 571–577.

[29] R. Millon, E. Frati, and E. Rucci, "A comparative study between HLS and HDL on SoC for image processing applications," 2020, *arXiv:2012.08320*.

[30] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network accelerator," 2017, *arXiv:1712.08934*.

[31] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2015, pp. 161–170.

[32] Y. Ma, N. Suda, Y. Cao, J.-S. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of convolutional neural networks onto FPGA," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–8.

[33] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–9.

[34] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 10, pp. 1217–1221, Oct. 2017.

[35] B. Khabbazan and S. Mirzakuchaki, "Design and implementation of a low-power, embedded CNN accelerator on a low-end FPGA," in *Proc. 22nd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2019, pp. 647–650.

[36] J. Vieira, R. P. Duarte, and H. C. Neto, "KNN-STUFF: KNN streaming unit for fpgas," *IEEE Access*, vol. 7, pp. 170864–170877, 2019.

[37] S. Gorgin, M. Gholamrezaei, D. Javaheri, and J.-A. Lee, "KNN-MSDF: A hardware accelerator for k-Nearest neighbors using most significant digit first computation," in *Proc. IEEE 35th Int. Syst. Chip Conf. (SOCC)*, Sep. 2022, pp. 1–6.

[38] D. Jamma, O. Ahmed, S. Areibi, and G. Grewal, "Hardware accelerators for the K-nearest neighbor algorithm using high level synthesis," in *Proc. 29th Int. Conf. Microelectron. (ICM)*, Dec. 2017, pp. 1–4.

[39] A. Kelati, H. Gaber, J. Plosila, and H. Tenhunen, "Implementation of K-nearest neighbor on field programmable gate arrays for appliance classification," in *Proc. IEEE 8th Int. Conf. Smart Energy Grid Eng. (SEGE)*, Aug. 2020, pp. 51–57.

[40] S. Vellas, G. Lentaris, K. Maragos, D. Soudris, Z. Kandylakis, and K. Karantzalos, "FPGA acceleration of hyperspectral image processing for high-speed detection applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[41] A. Zoulkarni, C. Kachris, and D. Soudris, "Hardware acceleration of decision tree learning algorithm," in *Proc. 9th Int. Conf. Modern Circuits Syst. Technol. (MOCAST)*, Sep. 2020, pp. 1–6.
[42] T. Tanaka, I. Ikeno, R. Tsuruoka, T. Kuchiba, W. Liao, and Y. Mitsuyama, "Development of autonomous driving system using programmable SoCs," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, Dec. 2019, pp. 453–456.
[43] C. Garcia, R. S. Tavares, A. D. Mora, J. Fonseca, H. Oliveira, and L. B. Oliveira, "FPGA-based satellite image classification for water bodies detection," in *Proc. Int. Young Engineers Forum (YEF-ECE)*, Jul. 2020, pp. 44–48.
[44] B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *J. Electron. Imag.*, vol. 13, no. 1, p. 146, Jan. 2004.
[45] R. Firdousi and S. Parveen, "Local thresholding techniques in image binarization," *Int. J. Eng. Comput. Sci.*, vol. 3, no. 3, pp. 4062–4065, 2014.
[46] P. Puneet and N. Garg, "Binarization techniques used for grey scale images," *Int. J. Comput. Appl.*, vol. 71, no. 1, pp. 8–11, Jun. 2013.
[47] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, and M. Nielsen, "Deep feature learning for knee cartilage segmentation using a tri-planar convolutional neural network," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* Cham, Switzerland: Springer, 2013, pp. 246–253.
[48] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 1237–1242.
[49] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Proc. Int. Conf. Artif. Neural Netw.* Berlin, Germany: Springer, Sep. 2010, pp. 92–101.
[50] S. Sağlam, F. Tat, and S. Bayar, "FPGA implementation of CNN algorithm for detecting malaria diseased blood cells," in *Proc. Int. Symp. Adv. Electr. Commun. Technol. (ISAECT)*, Nov. 2019, pp. 1–5.
[51] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, "Random forests and decision trees," *Int. J. Comput. Sci. Issues*, vol. 9, no. 5, p. 272, 2012.
[52] M. Lux and O. Marques, "Visual information retrieval using Java and LIRE," *Synth. Lectures Inf. Concepts, Retr., Services*, vol. 5, no. 1, pp. 1–112, Jan. 2013.

SERKAN SAGLAM received the B.S. and M.S. degrees in electric electronic engineering from Marmara University, Istanbul, Turkey, in 2018 and 2020, respectively. He is currently pursuing the Ph.D. degree in electric electronic engineering with the University of Southampton, Southampton, U.K. His Ph.D. research focuses on the intersection of image processing and deep learning, with a particular interest in the application of FPGA technology for enhancing the efficiency of parallel processing systems. He is exploring innovative approaches to optimize deep learning algorithms for real-time processing and is committed to contributing to advancements in hardware acceleration techniques.

SALIH BAYAR (Member, IEEE) received the B.S. degree in electronics and communication engineering from Yıldız Technical University, Istanbul, Turkey, in 2003, the M.S. degree in electrical engineering and information technology specialization in systems engineering from Karlsruhe Institute of Technology, Karlsruhe, Germany, in 2007, and the Ph.D. degree from the Department of Computer Engineering, Boğaziçi University, Istanbul. He was a Research Assistant with the Department of Computer Engineering, Boğaziçi University, from 2007 to 2013. He was a Research and Development Engineer and the Manager of a leading software company in Istanbul, from 2013 to 2017. Since 2017, he has been an Assistant Professor with the Electrical and Electronics Department, Marmara University, Istanbul. His research interests include parallel computing, machine learning, image processing, FPGAs, multi-processor, and embedded multi-core architectures.