IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 9, 2022, accepted May 19, 2022, date of publication May 25, 2022, date of current version June 6, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3177743

Instruction Set Extension of a RiscV Based SoC for Driver Drowsiness Detection

SEYED KIAN MOUSAVIKIA[1], ERFAN GHOLIZADEHAZARI[2], MORTEZA MOUSAZADEH[1], AND SIDDIKA BERNA ORS YALCIN[2]

[1]Department of Electrical Engineering, Faculty of Electrical and Computer Engineering, Urmia University, Urmia 5756151818, Iran
[2]Department of Electronics and Communication Engineering, Istanbul Technical University (ITU), Maslak, Istanbul 34469, Turkey

Corresponding author: Morteza Mousazadeh (m.mousazadeh@urmia.ac.ir)

This work was supported in part by the Scientific and Technological Research Council of Turkey (TUBITAK) and the Ministry of Science, Research and Technology of Iran (MSRT) under Project 119N641, and in part by Urmia University and Istanbul Technical University with the support of Tabriz University and the Center for International Scientific Cooperation of MSRT.

ABSTRACT This paper describes the design and implementation of a driver drowsiness detection (DDD) system using a modified RiscV processor on a field-programmable gate array (FPGA). To detect drowsiness, Convolutional Neural Network (CNN) is implemented on a RiscV processor. The CNN is trained to classify four primary driver's expressions, including distraction, natural, sleep, and yawn. The trained CNN accuracy is 81.07% on validation data. Furthermore, due to FPGA memory limitations, written C code for the trained CNN is optimized in numerous ways. Optimizations include the usage of dynamic fixed-point data types and dynamic memory allocations. On the other hand, the processor is modified by adding three custom instructions, including custom store, conv2d(2×2), and multiply and accumulation (MAC) to enhance the computation rate. As a result, the processor with custom store, conv2d(2×2), and MAC as custom instructions achieved the best result in terms of latency, with an improvement factor of 1.7 over the base processor and 1.25 over the processor with only custom store and multiply and accumulation (MAC) in exchange of slight increase in area.

INDEX TERMS Convolutional neural network, driver drowsiness detection, FPGA, hardware implementation, modified RiscV processor.

I. INTRODUCTION

Driver fatigue and drowsiness are the leading cause of human casualties in traffic accidents. Moreover, property damage is another concern for governments. To minimize these casualties, automotive companies have spent a tremendous amount of time and money to design systems that detect drowsiness and sleep and then alert the driver in these situations [1]. These systems must detect drowsiness before it leads to accidents; subsequently, the designed system must have high accuracy [2].

In recent years, neural networks, especially convolutional neural networks (CNNs), satisfy many requirements. These networks are noticed widely due to their outstanding functionality in classification with considerable accuracy [3]. Furthermore, these neural networks are implementable on the field-programmable gate arrays (FPGAs) by utilizing an embedded processor [4]. The embedded processor receives the camera's input, processes it, and finally illustrates the network's output on remarkably basic peripherals like seven segments or even light-emitting diodes (LEDs). In this paper, a CNN model is trained to detect driver drowsiness. Moreover, this trained model is implemented on the FPGA, which requires noticeable modification in the model parameters due to the shortage of resources in low-cost FPGAs. Even though shortage of resources on low-cost FPGA's can be challenging; still FPGA's are a great choice for hardware designers. The hardware designer can easily change the architecture of the designed system inside of the FPGA by adding or removing some lines of code, which is not possible on other hardware's like GPU's. Moreover, application-specific integrated circuit (ASIC) implementation out of FPGA is also possible.

Despite a wide range of research, many considered drowsiness the main parameter of car accidents, yet driver distraction leads to car accidents. Moreover, humans yawning

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

is a noticeable sign of sleepiness. As an immediate result, designing a system that can detect driver drowsiness is highly recommended. This paper proposed decreasing car accidents by considering driver drowsiness and distraction. The hardware architecture of our designed system is such that it can form a system on a chip (SoC) compatible with future ASIC implementation. Moreover, for the embedded processor instruction set architecture (ISA), RiscV is considered [5]. Due to its simplicity and being completely open-source, it is easily possible to modify and change the ISA to make the processor optimized for any application. According to the research firm Semico, the number of chips that include at least some RISC-V technology will grow 73.6 percent per year to 2027, when there will be some 25 billion AI chips produced, accounting for US $291 billion in revenue [6]. On the other hand, due to utilizing an embedded processor, the implementation of CNN becomes less challenging because of the ability to implement the designed CNN in C/C++. Additionally, codes in C/C++ can be compiled and translated into RiscV machine-level code due to a reliable RiscV GNU Tool-chain found in [7]. Optimizations on the software side are done by managing memory efficiently, as an instance in the compiler, appropriate linkers, and decreasing the number of parameters. Likewise, optimizations on the written code are considered due to the shortage of memory resources on low-cost FPGAs. These optimizations reduce the size of the machine-level code.

The organization of the paper is as follows. Section II provides information regarding previous research in designing driver drowsiness detection systems and RiscV processors. Section III describes the software design of the CNN and the dataset utilized for system training and validation. Moreover, the model's modifications for software optimizations on the implemented CNN are explained in this section. Section IV provides information concerning hardware implementation and optimizations. In Section V, results and achievements of the paper are illustrated. Finally, section VI concludes the paper.

II. RELATED WORK

By increasing the number of cars and trucks, the rate of car accidents increases regretfully. In these accidents, driver drowsiness has the most dominant role. Subsequently, researchers in both industry and academic endeavor to reduce the rate of car accidents by designing a system to detect drowsiness.

Between researches in the area of driver drowsiness detection, a significant number of them utilize biomedical methods to detect drowsiness. In [1], the authors provide valuable information about psychological signals that can detect a driver's drowsiness level. Authors also measured drowsiness levels utilizing various methods like Electroencephalography (EEG), Electrocardiography (ECG), Electrooculogram (EOG), and Electromyogram (EMG). The measured accuracy of some methods like EEG and ECG is above 95%. In [8] utilizing an EEG headset, brain activity is recorded, and via k-nearest neighbor and support vector machine (SVM) 95.8% and 93.8% accuracy were achieved. In [9], with ECG signal and the same classifiers of [8] driver drowsiness was detected, and above 90% accuracy was achieved for only two class classifications. In [10], the validity of detecting driver drowsiness by measuring the muscle activity associated with steering wheel grip with the help of EMG was analyzed. The results indicated that the validity of the proposed algorithm, needs to be more robustly tested with additional subjects and with different experimental designs. In [11], five methods for drowsiness detection, i.e. subjective reporting, driver biological features, driver physical features, vehicular features while driving, and hybrid features are discussed and compared, which indicates all methods have pros and cons, and even though some of the stated methods are very complex and need many sensors; they still need improvement. As a result, although the accuracy of these systems is considerably high in certain situations, the usage of numerous sensors sometimes simultaneously, and the increased complexity of the system are the main downsides of this method. Moreover, the cost of these methods is not affordable, and also the flexibility of modifying these systems to add other options is poor.

The other method for detecting drowsiness is image processing. These methods are great for high-level and software approaches since they are hard to implement on hardware without the help of high-level design languages such as Matlab or Python. In [12], an intelligent surveillance system is designed based on signal processing and embedded tools, which has three interlinked modules, driver fatigue detection, alcohol content detection, and vehicular crash detection. Although the system is complete and practical, the brain of the system is a Raspberry Pi microcomputer which is not flexible and far from optimized for such a huge task. In [13] however, an FPGA-based system for drowsiness detection is proposed. The system utilizes two signs to detect the driver's drowsiness, closed eye, and open mouth. Although the system is on FPGA, which means it can be designed to be optimum in terms of power and delay and suitable for ASIC implementations, the use of high-level design tools, especially the use of Matlab HDL coder, reduces the hardware designers' control on the final architecture. In [14], a real-time system that utilizes computerized camera to automatically track and process the driver's eye using Python, dlib, and OpenCV is proposed. With Python and high-level libraries like OpenCV and dlib, small bare-metal processors cannot host this method.

Another method that can be used for detecting drowsiness is neural networks especially convolutional neural networks (CNNs). In [15] DriCare, a driver drowsiness detection method is proposed by using face landmark detection and the help of CNN. Although the DriCare method accuracy is about 92%, but the system is tested on an Intel Core i7 CPU, which is a supercomputer in comparison with small embedded processors. In [16], a real-time model based on deep neural networks on an embedded processor is proposed.

In this paper, utilizing minimal facial landmarks, accuracy is about 89.5% for three-class classification, and a detection speed is 14.9 frames per second (FPS) tested on Jetson TK1. Although the network is modified for an embedded processor, nothing about the hardware optimization is proposed since the hardware is fixed and not flexible. In [17], a similar approach using multi-task CNN (MTCNN) is proposed. Eye and mouth characteristics are utilized for the driver's behavior model. Changes to these characteristics are used to monitor driver fatigue. Again the accuracy is high (98%) on the proposed datasets, but no discussion about hardware is made.

In this paper, we decided to utilize neural networks to detect drowsiness since image processing techniques are very heavy to implement on embedded processors. Also, the use of biomedical techniques can result in an accurate system in some situations, but not affordable.

As a hardware-software codesign matter, the selected method must run on suitable hardware. In recent years RiscV has gained popularity due to its open-source nature. For deep learning algorithms, however, the majority of research indicates that the designers paired a base CPU with an accelerator to decrease the latency of the implemented neural network. In [18], the authors proposed a RiscV-based hardware accelerator designed for the Yolo object detection system. The designed system executed Yolo in 400ms. In [19], the authors compared two hardware accelerators, NVDLA and Gemmini, and they proved that NVIDIA's NVDLA accelerator outperforms Gemmini by 3.77x running ResNet-50 on an equivalent configuration using the same system setup. However, in [20], the author mentioned that a high-end FPGA is required to test the NVIDIA's NVDLA accelerator. As a result, they proposed to integrate NVDLA into a real RiscV SoC on the Amazon cloud FPGA using a tool named FireSim. As a result, although adding an accelerator to a base CPU is logical, but the added area to the base processor requires higher-end FPGAs and, as a result bigger chip which all leads to higher costs. Moreover, the RiscV ISA is open, and the designer can add nearly as many new instructions as is needed to make the processor customized. In [21], a Laplacian filter SoC with RiscV processor over wishbone protocol implemented. As a result, hardware implementation by adding a single DSP block was 36 times better than software implementation of Laplacian filter. Consequently, we tried to implement a very efficient CNN regarding the number of parameters on a modified RiscV CPU.

III. DATASET AND CNN MODEL

In this section, we propose the dataset, and our designed CNN models. The dataset is described in III-A, and the CNN models are described in III-B. In this paper, the classified images depict various behaviors of drivers. In this case, drivers yawn, sleep, and distraction are considered as criteria that increase the probability of car accidents.

A. DATASET

In this paper, we consider several criteria which lead to car accidents. While driver drowsiness is pivotal, considering driver distraction is essential as well. Consequently, we designed the system by considering driver drowsiness to detect yawning and sleeping. Furthermore, driver distraction is considered. Despite numerous methods based on image processing and machine vision, a dataset that addresses our requirements is not supplied. Subsequently, in this paper, frames of the videos from reference [22] are extracted to design an accurate system. The frames of these videos are extracted at the rate of 1 frame per second, then the frames are labeled in four distinct classes, including normal, distraction, yawn, and sleep. After the initial design with the extracted dataset, several images were added to the dataset to increase the system's accuracy. In this paper, we consider a scenario where the camera is fixed on the car's mirror. Fig. 1 (a) to (d), illustrate a sample of distinct classes, including normal, distraction, sleep, and yawn, respectively.

For these classes, intending to improve the system accuracy and sensitivity, extracted frames are augmented by adding Gaussian noise to the images, changing the brightness of images, translation, and rotation, as can be seen in Fig. 1 (e) to (h). It must be noted that the system quality and received images might be affected due to driving cars in tunnels, bumpy roads, and the like. Consequently, we consider Gaussian noise with a wide range of σ, including 0.01, 0.02, and 0.04. The reason for considering noises is due to the various quality of the camera and changing the quality of input images. Furthermore, while cars are derived at different times, the light varies considerably. For this reason, to improve the system sensitivity, the effect of the light is considered. It must be noted that, due to the mobility of cars, moving in various directions, and the hills and valleys on the roads, the driver's location, and position extracted by the inputs of the camera witness conspicuous changes. Consequently, the system cannot detect the correct position of the drivers. For this reason, in this paper, we consider the rotation of the images and translation. Overall, by considering these criteria for data augmentation, the number of samples for normal, distraction, yawn, and sleep are 5069, 5063, 5062, and 5109.

B. CNN MODEL

Driver drowsiness detection systems have been used widely due to their profound effects on society. In this paper, the system is implemented on both sides; software, and hardware. It must be noted that for designing the CNN, some limitations must be considered related to the hardware shortage memory. Furthermore, while the number of parameters of the CNN has profound effects on the system accuracy, as the driver drowsiness detection system must be real-time, the system must be lightweight. The implemented CNN for this system is presented by Model Summary 1, which has 3588 parameters:

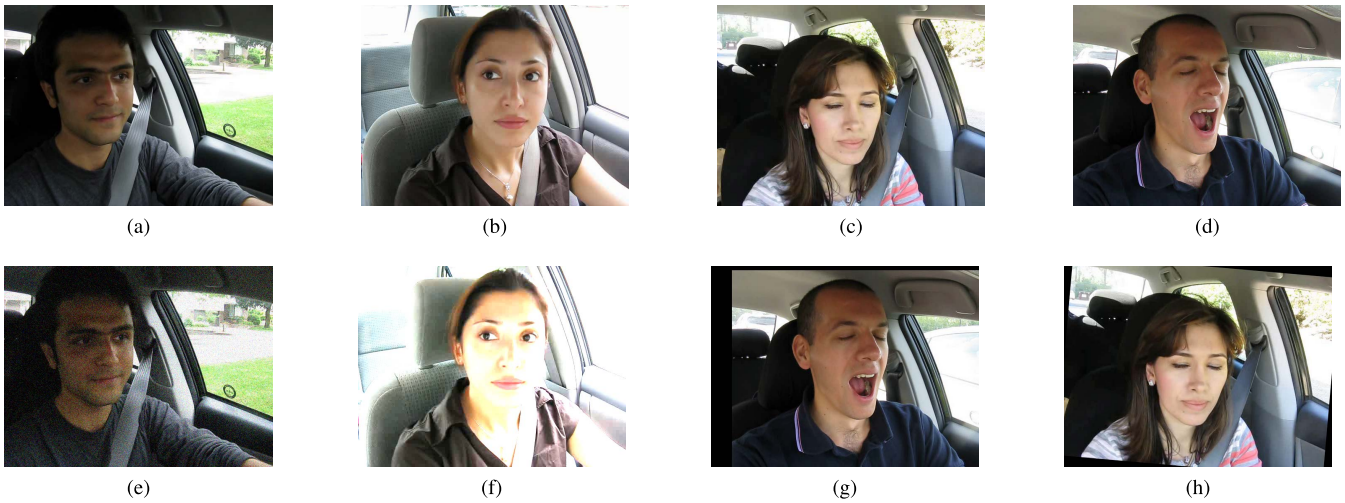

FIGURE 1. Extracted frames of driver expressions (a) Normal, (b) Distracted, (c) Sleep, (d) Yawn, (e) Gaussian Noise, (f) Brighter, (g) Translation, (h) Rotation.

Model Summary 1: Implemented CNN Structure
Conv2D(4, (2,2), *input shape* = (100,100,1), activation = 'relu')
MaxPooling2D (MaxPooling2D (2,2))
Conv2D(4, (2,2), activation = 'relu')
MaxPooling2D(*pool size* = (2,2))
Conv2D(4, (2,2), activation = 'relu')
MaxPooling2D(*pool size* = (2,2))
Conv2D(4, (2,2), activation = 'relu')
MaxPooling2D(*pool size* = (2,2))
Flatten()
Dense(32, activation = 'relu')
Dense(classes, activation = 'softmax')

As illustrated by Model Summary 1, the system is extremely lightweight and has a low number of parameters. Input image size is 100 × 100, and images for training and testing are grayscale. It must be noted that the CNN model is designed based on the hardware limitations. For implementing the model on the hardware, the input size of the image and the number of filters alongside the size of the filters in the first convolutional layer's have a significant effect on the area of the implemented model on hardware. Therefore, memory usage is managed to use the minimum amount due to a lack of memory on low-cost FPGAs. Besides the input image and convolutional layers, the activation function of all convolution layers is considered to be rectified linear unite (RELU) in this model. Regarding the max-pooling layers, stride size is two, and padding does not add. There are also two fully connected layers at the end of the model, first with RELU activation and second with SoftMax activation, as four classes are considered. After training the model on the described dataset, the achieved accuracy for validation data is 81.07%. Fig. 2 (c) and Fig. 2 (f) depicts the system accuracy and loss in terms of the number of epochs for the CNN model.

In this paper, due to the hardware limitations, the designed CNN is compact. The implemented CNN's layers are listed in Model Summary 1. It must be noted that in the system utilized for driver drowsiness detection, the system accuracy must be as accurate as possible. For this reason, two additional models, CNN I and CNN II, are designed for having a system with high accuracy. These models are implementable like Model Summary 1, but they require more memory. The CNN II model is depicted in Fig. 3. Regarding CNN I, the first two convolution layers have three channels, similar to CNN II. However, the remaining convolution layers in CNN I have 4, 8, 16, and 32 channels. After each two convolution layers, a max-pooling layer is considered. Additionally, the first dense layer has 128 neurons. Furthermore, the size of input images in CNN I and CNN II is 160 × 120, then the number of parameters in CNN I and CNN II are 21,282 and 45,546, respectively. Moreover, CNN I and CNN II accuracy and loss functions are depicted in Fig. 2.

Intending to evaluate the proposed models, we trained several well-known models with our dataset, including MobileNet V2, VGG-16, and Inception. While for Model Summary 1, we trained the network with grayscale images, this time, the networks are trained with an RGB version of our dataset. In table 1 the CNN II accuracy and its number of parameters are compared with famous state-of-the-art networks. However, the reason for training with the RGB data is that the models MobileNet V2, VGG-16, and Inception are trained with the ImageNet dataset, which contains RGB data, and the weights are extracted for RGB images. As a result, for having a fair comparison between CNN II, and the pre-trained models, CNN II is trained by RGB data as well. While the accuracy of MobileNet is significantly higher than that of CNN II, the accuracy of VGG-16 is approximately the same as CNN II trained with RGB data. Moreover, the accuracy of Inception is lower than CNN II. It must be noted that the number of parameters in this pre-trained model is by far more than the CNN II. These results indicate that although CNN I

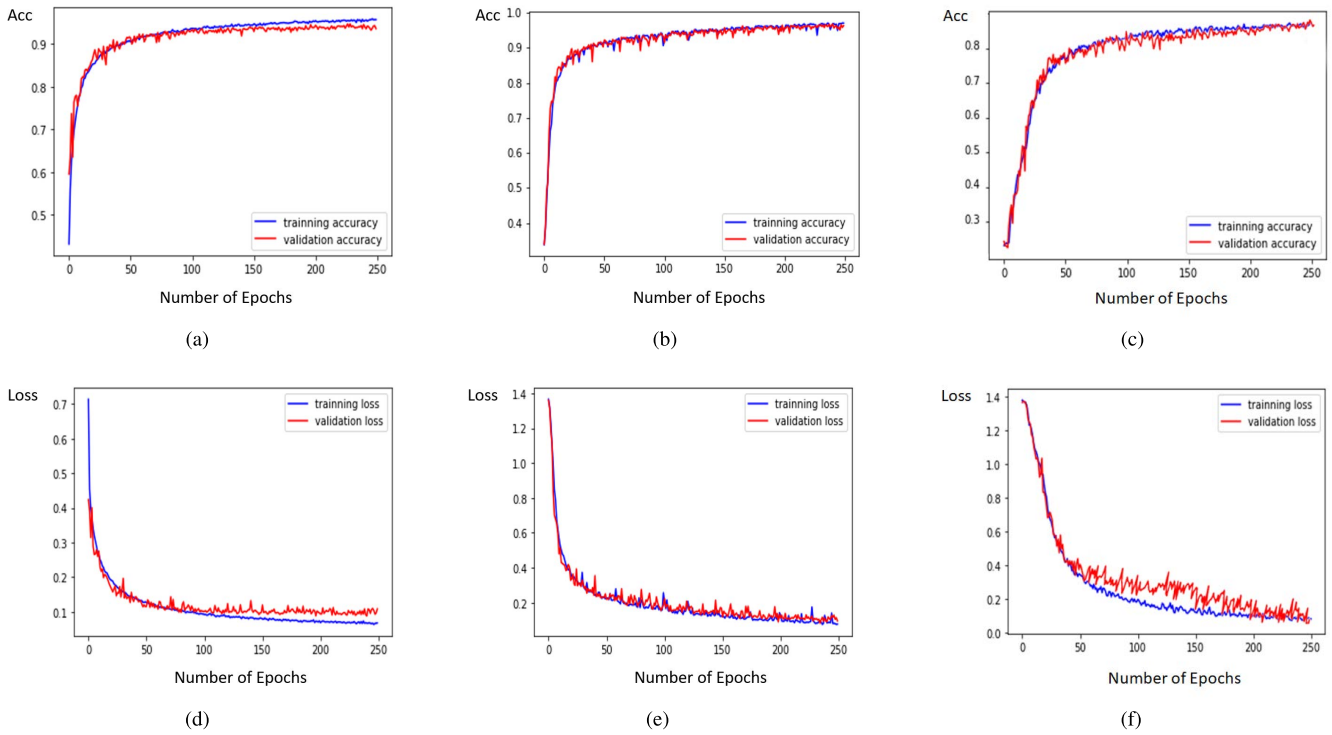

FIGURE 2. (a): CNN I Accuracy, (b): CNN II Accuracy, (c): Implemented CNN Accuracy, (d): CNN I Loss, (e): CNN II Loss, (f): Implemented CNN Loss.

and CNN II have more parameters than Model Summary 1, but they are a much better choices than other networks like VGG-16. Furthermore, if a bigger FPGA is available, CNN I and CNN II are better candidates than networks like VGG-16, Inception, and even MobileNet V2.

Also, we should mention that we trained all of the above models with Python library, Keras, optimizer was Adam, and loss function was cross-entropy. Furthermore, we extracted the weights and biases of the Model Summary 1 for C/C++ implementation. In addition to designing the model with a low number of parameters, the format of the weights and biases are modified as well. The reasons and results of modifications are explained in subsection III-C.

C. SOFTWARE OPTIMIZATION

This subsection covers optimizations considered in writing C code to achieve minimum code size for the implemented CNN. As we mentioned in subsection III-B, the lack of memory in low-cost FPGA models leads to selecting the CNN model, which has a low number of weights and biases. Despite choosing a lightweight model, tackling these issues requires even more consideration. Overall, for some systems, especially memory-hungry systems, the memory unit on FPGAs might be insufficient. For this reason, intending to decrease the needed memory block size on FPGA, all of the weights and biases are converted into fixed-point numbers. For conversion from floating-point to fixed-point, a python package named fxpmath is utilized [23]. Also, for a better understanding of the fixed-point configuration in our design

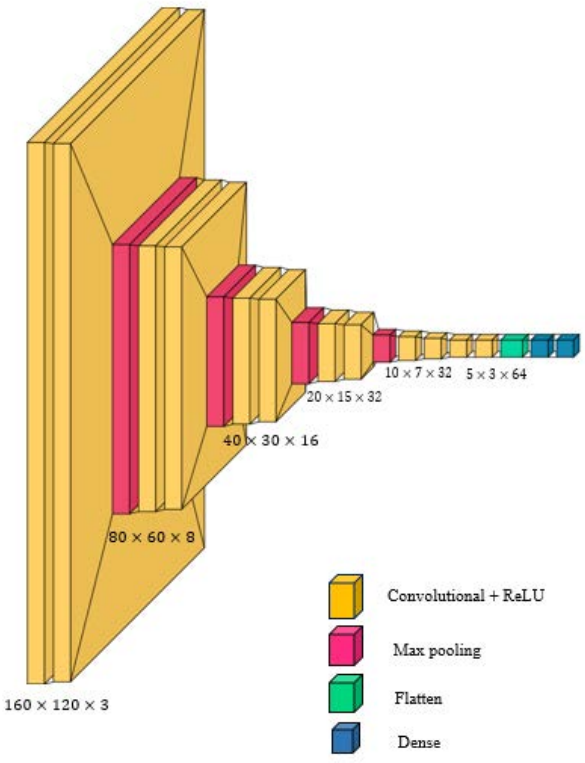

FIGURE 3. The structure design of The improved CNN II model.

and comparing it to the famous IEEE 754 standard for float numbers in Fig. 4, our fixed-point configuration can be seen.

TABLE 1. CNN II accuracy and well-known models accuracy.

Dataset Description	Gaussian Noise and Brightness	Gaussian Noise and Brightness	Gaussian Noise and Brightness	Gaussian Noise and Brightness	Gaussian Noise and Brightness
Network Description	CNN II - Grayscale	CNN II - RGB	Mobile Net V2	VGG-16	Inception
Number of Parameters	45,546	45,570	2,557,984	18,649,412	34,390,820
System Accuracy	97.88%	95.29%	99.26%	95.91%	94.75%

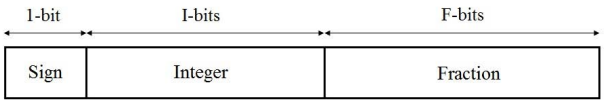

FIGURE 4. Configuration of utilized fixed-point number.

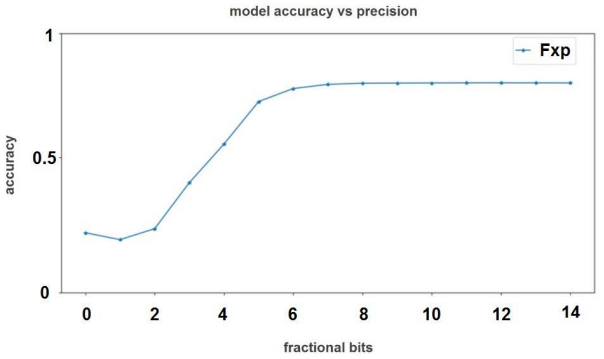

FIGURE 5. Accuracy evaluation of implemented CNN (Integer part is equal 8 bits including 1 bit for sign).

Unlike hardware description languages like Verilog or Very High-Speed Integrated Circuit Hardware Description Language (VHDL), in C or C++ programming languages, only valid data types are 8bits, 16bits, 32bits, and 64bits, so there are not many options for demonstrating network parameters. As a result, 8bits for network parameters and 16bits for intermediate variables are considered to reduce the compiled code size compared to int(32bits) and double(64bits) in C language programming, respectively. Nevertheless, utilizing this modification requires more consideration to prevent a considerable drop in the system accuracy. It is undeniable that using 32bit float data types for computation and demonstration of weights and biases in the system leads to the same accuracy achieved in the training phase; compared to the situation, the number of digits for saving network parameters and computation is decreased. However, by utilizing fxpmath, the system accuracy is recalculated. We first tested the system accuracy with a wide range of fraction bits (1 to 15 bits), changing just network parameters. Fig. 5 demonstrates system accuracy while a wide range of bits for fraction part is utilized. Note that the integer part of the number is set to 8 bits. This test demonstrates that for network parameters, the number of fraction bits (F bits) has the most importance to systems accuracy.

As mentioned, in C/C++, data types are limited. As a result, the second test was considered for finding the best configuration for 8bit fixed-point representation. In this test, the whole number is 8bits ($S + I + F$), and the fraction part is changed from 0 to 8 for finding the best configuration for the network parameters. Table 2 indicates that 6 bits for the fraction part and 1 bit for integer, including 1 bit for the sign ($I = 2, F = 6$) results in 73.7% accuracy, which is the closest accuracy to 81.07% achieved in validation accuracy with float data types.

TABLE 2. Accuracy evaluation of implemented CNN with fixed-point parameters (whole number is 8 bits).

Fraction Bits (F)	Accuracy
0 bit	23.2%
1 bit	23.1%
2 bits	20.5%
3 bits	24.7%
4 bits	42.4%
5 bits	57.3%
6 bits	73.3%
7 bits	44.8%
8 bits	29.6%

For other variables such as the input of the network and the intermediate variables, fixed-point configuration must be decided as well. Since in the training phase, the input of the CNN is normalized between 0 and 1; as a result, all of the 8 bits can be dedicated to the fraction part. However, the utilized camera gives 565 output (5 bits for red, 6 bits for green, and 5 bits for blue). As a result, converting the input picture to grayscale, all three channels must be added, leading to a maximum of 7 bits. Subsequently, the input layer configuration is selected to be 7 bits for the fraction part, leading 1 bit for the sign, which is always 0. For intermediate variables after multiplying two 8-bit numbers, the result is 16 bits. These 16 bits results are reduced to 8 bits after all the calculations of the layer are done. With this method, both precision and memory shortage can be satisfied. However, for reducing 16 bits to 8 bits, the amount of right shift decides the configuration of the 8-bit number. This configuration must change throughout the network since, at the first layers, numbers are small and close (more fraction bits are needed), but at the final layers, numbers grow and become widespread, so more bit for the integer part is needed. Fig. 6 supports this idea and depicts the output range of each layer of our implemented CNN for all of the pictures of our dataset. This decision results in a dynamic fixed-point configuration. Table 3 depicts the configuration of each layer's output, which is deduced from Fig. 6. Note that all weights have the configuration of 1 bit for sign, 1 bit for integer, and 6 bits for fraction part resulting from Table 2.

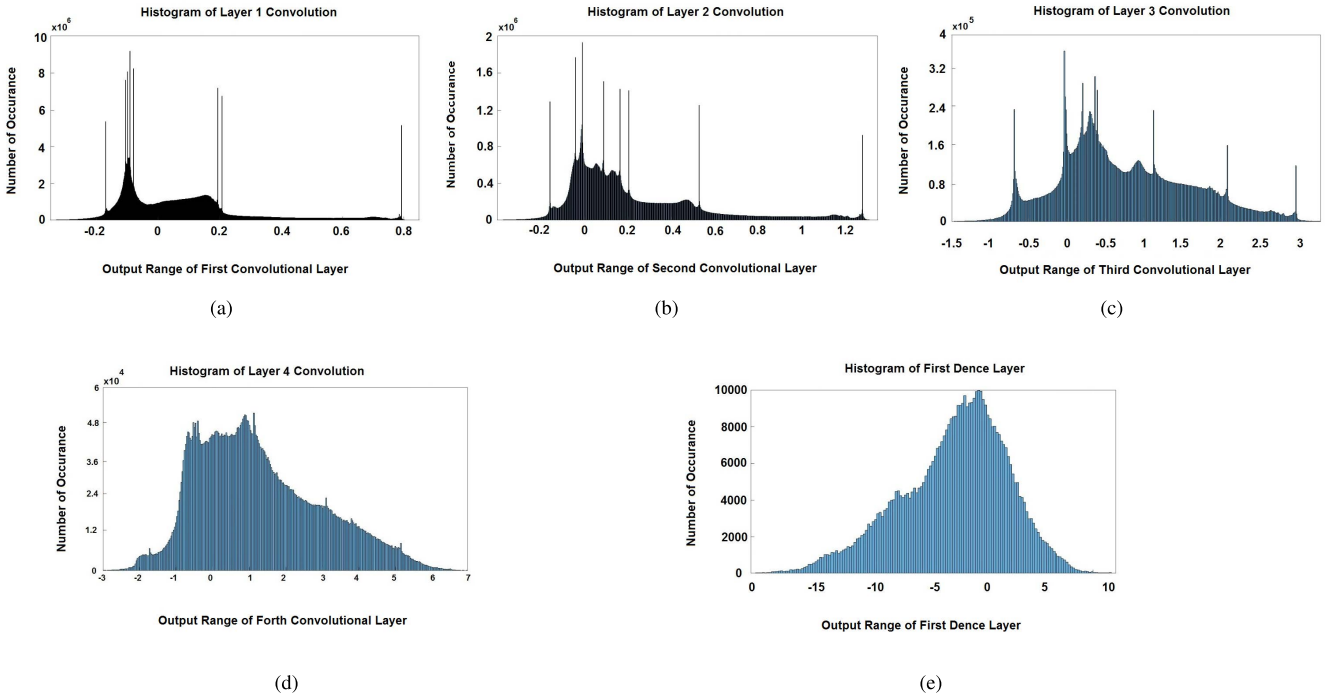

FIGURE 6. Output histogram of (a) First, (b) Second, (c) Third, and (d) Forth convolution layer, (e) First dense layer.

TABLE 3. Fixed-point configuration of each layer's output.

Layer	Output Configuration
Convolution 1	$S = 1$ bit, $I = 0$ bit, $F = 7$ bits
Convolution 2	$S = 1$ bit, $I = 1$ bit, $F = 6$ bits
Convolution 3	$S = 1$ bit, $I = 2$ bits, $F = 5$ bits
Convolution 4	$S = 1$ bit, $I = 3$ bits, $F = 4$ bits
Dense 1	$S = 1$ bit, $I = 4$ bits, $F = 3$ bits

Finally, for the last dense layers configuration, we decided to keep 16-bit representation, to better differentiate between four classes of the network.

For the second optimization, due to an increase in the number of intermediate variables after each convolution layer, we utilized Dynamic Memory Allocation (DMA) functions, especially *Malloc*() and *free*() in coding. By these functions, the heap memory of the processor is allocated and deallocated to the variables dynamically. For example, by these functions, after calculating the first layer's output, there is no need for the first layer's input to occupy the memory anymore. Subsequently, the occupied memory is released with the *free*() function and automatically reallocated to the following layer's variables. However, due to the architecture of the RiscV compiler, usage of dynamic memory allocation functions in C or C++ is not possible unless the target embedded processor has an operating system. The $-libgloss$ library contains these functions, but it applies to processors with an operating system. In this case, we use a Bare-Metal processor on FPGA, so the usage of $-libgloss$ must be avoided in the *Makefile* of the compiler. Then a specific version of the $-sbrk()$ library for the Bare-Metal systems compiler is considered to implement dynamic memory allocation. Fig. 7 provides a block diagram to indicate how to compile C codes that contain DMA functions for Bare-Metal cpu's. Also, in Fig. 7, we should mention that the $-sbrk()$ must be added at the first line of the written C code before other functions to allow the compiler to utilize DMA functions. Moreover, we endeavored to use points stated in [24] to optimize the written *Makefile* and *linkerfile* for the compiler.

Alongside the optimizations mentioned above for memory usage, we optimized the code in terms of run time and latency with the help of custom instructions. Overall, the convolution function is the most called in this code. Algorithm 2 depicts the pseudo-code of a standard convolution which indicates that the multiply and accumulation (MAC) operation frequently happens inside for loops. Due to this reason, it was convenient to add MAC operation or even the whole convolution operation to the hardware. Besides, in the case of MAC operation, it can also be used for two fully connected layers of our implemented CNN network to reduce the run time of the code even more. In Section IV, the hardware optimizations are explained by considering the process of adding a custom store as custom0, conv2d(2×2) operation as custom1, and MAC as custom2 to the Arithmetic Logic Unit (ALU) and processor decoder. However, in C code optimization, Fig. 8 displays a block diagram of how to modify the C code and the compiler to utilize the help of custom instructions. It must be noted that for conv2d(2×2) operation as custom1, considering the 8bit demonstration and comprehending the size of the convolution filters is 2×2, only the first operand of the 32bit ALU is

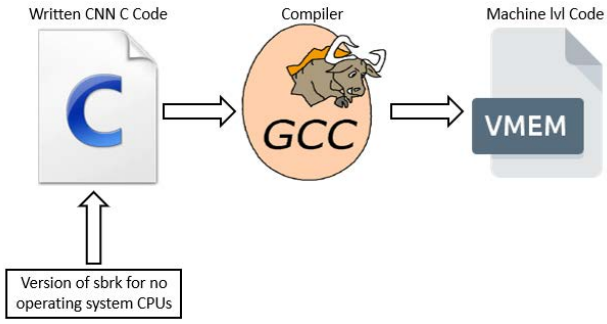

FIGURE 7. Compiling codes containing DMA functions for bare-metal CPU's.

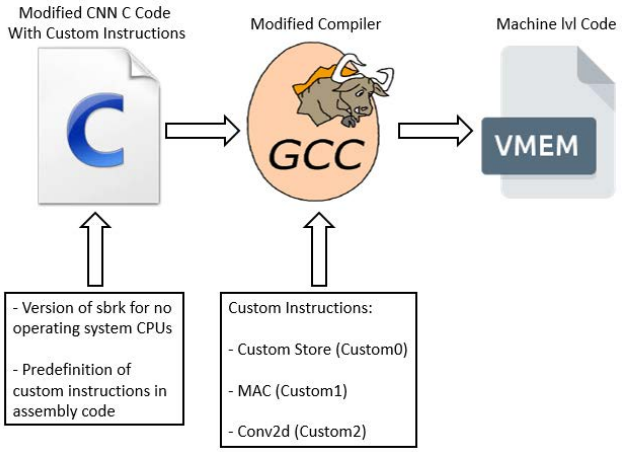

FIGURE 8. Compiling codes containing DMA and custom instruction functions for bare-metal CPU's.

occupied. The second unused operand can be utilized if a 16bit representation is used.

Regarding the SoftMax activation function used in the last layer of almost all CNN's, instead of using (1), where i is the target class; in the code, we compare outputs of the final fully-connected layer and picked the index of the most enormous number as the predicted class of the input picture. This approach reduces run time and resource utilization since for the processor, executing simple if statements are much more straightforward than executing complex equations like (1).

Finally, for better deducing the driver's primary expression and neutralizing unnecessary alarms, instead of just taking one picture and then classifying it; based on frame rate, the system classifies frames for 15 seconds, and after this time is elapsed based on what class was selected more often, then the driver's expression is deduced. This method is used in all of our implementations.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{\#classes} e^{x_i}} \tag{1}$$

IV. HARDWARE IMPLEMENTATION AND OPTIMIZATION

This section describes hardware implementation and optimizations of the trained CNN. As for the embedded

Algorithm 2: Normal Convolution

```
for i from 0 to (InputFeatureMapRowSize - 1)
  for j from 0 to (InputFeatureMapColumnSize - 1)
    oc[i][j] = 0
    for k from 0 to (WeightMatrixRowSize - 1)
      for l from 0 to (WeightMatrixColumnSize - 1)
        oc[i][j] = oc[i][j] + w[k][l] * in[i+k][j+l]
      end for
    end for
  end for
end for
```

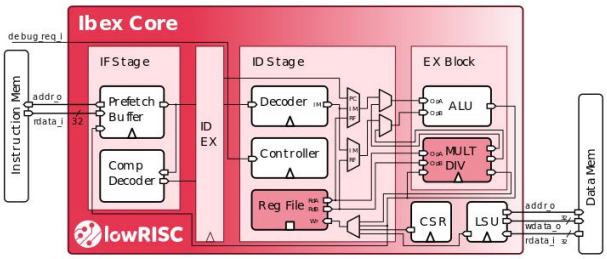

FIGURE 9. Block diagram of ibex core.

processor, we utilized the Ibex core [25]. Ibex is an RV32IMC, written in System Verilog, with two pipeline stages, as shown in Fig. 9. The two stages of ibex core are: instruction fetch(IF), instruction decoder(ID), and execution block(EX), which are merged together [25]. First, instructions are fetched into a prefetch buffer. Then instruction fetch controller supplies new instructions and their program counter to the ID-EX stage. After that, the ID-EX stage takes instructions and data from the IF stage, decodes them, and executes the instructions. This stage is made up of multiple sub-blocks such as controller, decoder, register file, arithmetic logic unit (ALU), multiplier/divider block (MULT/DIV), control and status register block (CSR), and load-store Unit (LSU) as shown in Fig. 9.

We utilized Ibex core as the central processor because of its simplicity and easy-to-understand architecture. Besides these reasons, the Ibex core is small enough to fit nearly on any low-cost FPGAs. Other more complicated RiscV cores such as Berkley's Rocket or BOOM found in the Chipyard repository [26] can also be tested in future designs.

Additionally, two remaining building blocks of our design are a camera module and a monitor. These modules can communicate to IBEX through the designed Wishbone B4 interface [27]. The choice of Wishbone over other interfaces like AXI and AMBA [28] is because the Wishbone interface is simple and open source, like IBEX and RiscV ISA. Moreover, in our designed wishbone protocol, all masters and slaves communicate through a shared bus called interconnect with an arbiter register controlling masters priority. Furthermore, slaves are selected according to their base addresses, which masters apply. Subsequently, first,

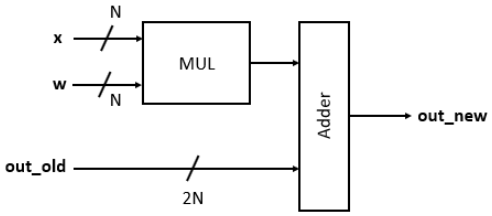

FIGURE 10. One MAC block.

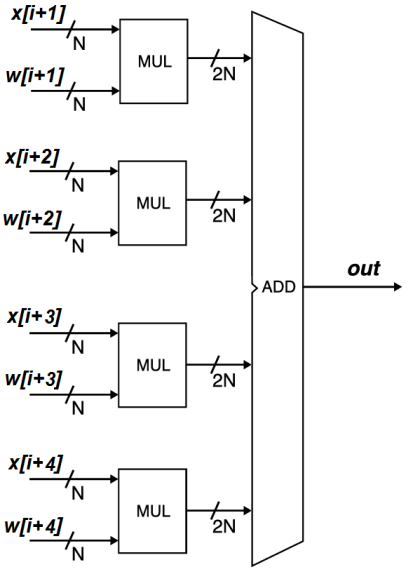

FIGURE 11. Conv2d(2 × 2).

the camera IP becomes active and populates the RAM with pixels. Next, the Ibex core instruction & data memory get active and process the image with the written neural network code inside the processor. Finally, the chosen class will be demonstrated on LED's. The overall structure of our system with added custom instruction blocks is depicted in Fig. 13. In this figure, the VGA block is optional and can be omitted without affecting the system. Also, a built-in timer inside the processor is used in the evaluation mode for measuring the processing time in hardware implementation.

As mentioned, the most called function in our written C code is the convolution function. In this function, the most frequent operation is MAC. Besides the convolution function, MAC operation is also used in fully connected layers. MAC operation in our C code happens inside for loops, as noted in Algorithm. 2. As a result, one MAC operation can be added to the processor (Fig. 10). Moreover, for the convolution function, since 2×2 convolutions are used in our design, four MAC operations can be combined and operate in parallel lanes to form one complete convolution (Fig. 11). This instruction is called conv2d(2×2) since we have two-dimensional convolutions with 2×2 filters in our implemented CNN network.

In this way, each time the data iterates through for loop, we gain a speedup relative to the method we use, and this speedup is escalated each time the code calls a

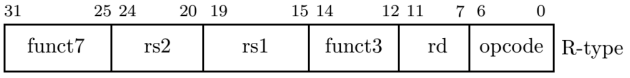

FIGURE 12. R-type instruction format.

convolution function. However, the only downside of adding the mentioned custom instructions is the overhead of resource utilization in FPGA. Fortunately, this increase is negligible for both cases, as we will discuss the results of adding these instructions to hardware in the following section. As a result, three custom instructions, one for storing kernel weights, one for MAC operation that can be used in convolution and fully connected layers, and one for whole convolution operation, are added to the decoder and ALU stage of the Ibex processor. Furthermore, these instructions read the rs1-register and rs2-register as source operands and write back the result into rd register like other R-Type instructions in RiscV ISA [5]. Moreover, the type of operation selection is done by funct7 and funct3 as shown in Fig. 12.

It is essential to note that MAC and convolution operation both happen in a series of the first store instruction, requiring previously stored data for operation. Besides adding this instructions to the hardware of the core, RiscV GNU Tool-chain must be recompiled to recognize the related instructions for newly added custom instruction opcodes.

The other optimization we considered for lowering memory size is to make the picture size received from the camera to the size that CNN is trained for. In camera modules, however, the usual size of the output picture is VGA (640×480), but its fractions like QVGA (320×240) and QQVGA (160×120) are also possible to achieve. On the other hand, it is possible to adapt the image size from the camera inside the C code by either cropping the image or manually resizing it. Since cropping can worsen the accuracy because of losing essential parts of the picture, resizing is preferred. Moreover, in our design, the first resizing is handled inside hardware by taking one of every two pixels coming from the camera. While the camera supports standard QVGA (320×240), its output images are resized to the QQVGA (160×120). Using this trick in hardware resulted in a 14% reduction of BRAM's size on the utilized FPGA board with no added code in the software. The second conversion, cropping, happened inside C code, which converts QQVGA to 100×100 suitable for the implemented CNN. This step does not require if the CNN's input is a fraction of the VGA like CNN1 and CNN2 networks, stated in III-B.

A. RASPBERRY PI IMPLEMENTATION

This subsection covers Raspberry Pi's implementation of the designed CNN. Since implementing the designed CNN's on FPGA is very time-consuming; it is recommended first to implement them on Raspberry Pi to test the system's accuracy. Fig. 14 depicts our driver drowsiness detection system utilizing a Raspberry Pi board. In this system, four

TABLE 4. Comparison between custom instructions.

Processor	Accuracy%	Frame rate (FPS)	Frame rate Improvement
Base Processor	63%	2.56 fps	1
With MAC Instruction	63%	3.46 fps	1.35
With conv2d(2 × 2) & MAC Instruction	63%	4.33 fps	1.7

TABLE 5. Board utilization comparison between processors.

Processor	LUT	FF	BRAM	DSP	IO	BUFG	PLL
Base Processor	9.15%	1.77%	99.26%	0.42%	30.48%	18.75%	16.67%
With MAC Instruction	9.39%	1.77%	99.26%	1.67%	30.48%	18.75%	16.67%
With conv2d(2 × 2) & MAC	9.57%	1.77%	99.26%	4.17%	30.48%	18.75%	16.67%

TABLE 6. Raspberry Pi and FPGA implementation comparison.

Implementation Method	Data Type	Accuracy	Clock Frequency	Frame Rate
Raspberry Pi	Float32	68%	1.2 GHz	14.28 fps
FPGA (Base Processor)	Dynamic Fixed-Point	63%	50 MHz	2.56 fps
FPGA (with MAC)	Dynamic Fixed-Point	63%	50 MHz	3.46 fps
FPGA (With conv2d(2 × 2) & MAC)	Dynamic Fixed-Point	63%	50 MHz	4.33 fps

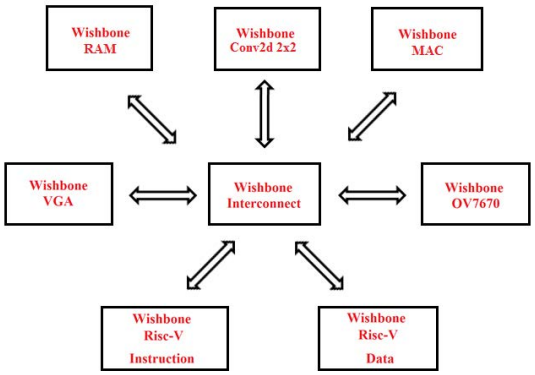

FIGURE 13. Overall system.

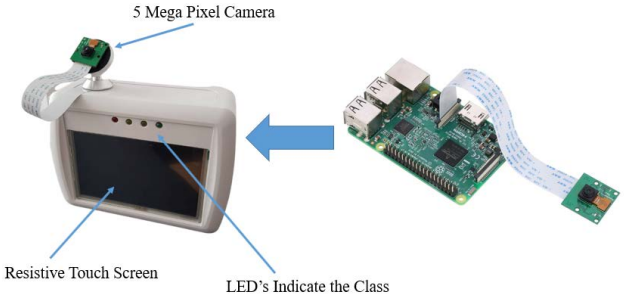

FIGURE 14. Designed system with raspberry pi board.

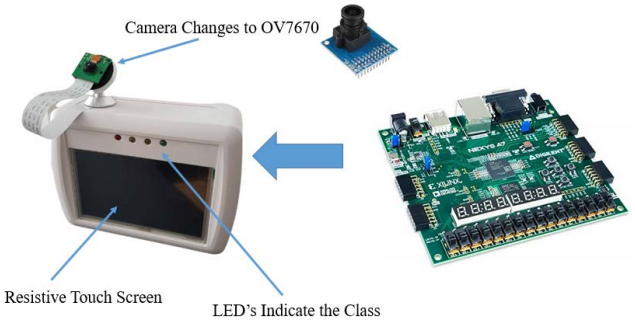

FIGURE 15. Designed system with FPGA (Nexys 4 DDR) board.

LED's illustrate the current expression of the driver based on the received picture. Also, a 5-inch HDMI resistive touch screen is provided to demonstrate the pictures taken from the camera and form a GUI (Graphical User Interface) for the user.

Furthermore, implementing the designed CNN's is achieved via the MATLAB support package for Raspberry Pi hardware [29]. Although all CNN models are trained on the Keras framework, utilizing the NVIDIA Tesla K80 Accelerator, the CNN model *.h*5 file is converted to DAG or series network format with the help of MATLAB Deep Learning Toolbox Converter for TensorFlow models found in Add-Ons section of MATLAB [30]. In this way, MATLAB compiles the CNN model to the ARM-based processor core of Raspberry Pi. However, MATLAB implements the networks with IEEE754 standard, and also, on the Raspberry Pi, the utilized camera is five megapixels instead of 0.3 megapixels of OV7670 used on FPGA design. As a result, the implemented design on Raspberry Pi will be the best-case scenario and can achieve better results in terms of accuracy compared to the same network implemented on FPGA. Besides accuracy, the frame rate achieved by Raspberry Pi can also be higher due to a much faster 1.2GHz clock frequency and the fact of having multiple cores on Raspberry Pi compared to a 50MHz clock and single core of ibex.

V. RESULTS

In this section, the results of the designed system are demonstrated. For hardware implementation on FPGA, Nexys 4 DDR FPGA board [31] and OV7670 camera module

FIGURE 16. Real word test of designed systems.

are used. For illustrating the system output and captured images from the camera module, an HDMI monitor used in Fig. 14 is connected to VGA connectors of the FPGA board via VGA to HDMI converter 15. However, the monitor block is optional and can be omitted when implementing the real-world application. For latency evaluation, we enabled a counter at the beginning of the C code and disabled it at the end of the code to measure the run time of each code precisely. As a result of an example picture from the dataset, latency is 390ms for ibex with no custom instruction, 289ms for ibex with custom store and MAC used in convolution and fully connected function, and 231ms for ibex with custom store, conv2d(2×2) for convolution and MAC for fully connected function as shown in Table 4. For accuracy measurements, we tested both FPGA and Raspberry Pi based systems in real-world and deployed the system inside a car, as can be seen in Fig. 16. Moreover, the accuracy results can be seen in Tables 4 and 6. Besides for FPGA implementation, the Vivado utilization table is considered to compare resource utilization in both cases with and without custom instructions, as shown in Table 5. Finally, Table 6 compares the FPGA implementation with Raspberry Pi implementation.

Overall, results indicate that the processor with added custom store, conv2d(2×2), and MAC operation can achieve the best result in terms of latency but with negligible increase in the usage of DSP blocks of the board.

VI. CONCLUSION

This paper discusses the implementation of a modified embedded processor based on RiscV ISA for driver drowsiness detection systems. The whole system consists of an embedded processor, camera, and a monitor connected to the VGA port of the FPGA. The drowsiness is detected with a convolutional neural network. Implemented CNN classifies input images taken from the driver into four classes: distraction, natural, sleep, and yawning. Moreover, system's hardware and software parts are optimized for this application. On the software side, Dynamic Memory Allocation and the use of dynamic fixed-point numbers instead of floating-point numbers for the weight, biases, and intermediate variables of the CNN are used. On the hardware side, three custom instructions, including a custom store, conv2d(2×2), and multiply and accumulation (MAC) operations, are added to the processor's ISA, decoder, and ALU sections to decrease the run-time of the code. Also, the camera module's taken picture size is changed from QVGA to QQVGA, intending to save more memory for CNN. As a result, by adding a custom store, MAC, and conv2d(2×2) as custom instructions, the improvement factor over the base processor is 1.7, which is higher than the improvement factor achieved by adding custom store and MAC as custom instructions which is 1.35.

REFERENCES

[1] A. Chowdhury, R. Shankaran, M. Kavakli, and M. M. Haque, "Sensor applications and physiological features in drivers' drowsiness detection: A review," *IEEE Sensors J.*, vol. 18, no. 8, pp. 3055–3067, Apr. 2018.

[2] A. Quddus, A. S. Zandi, L. Prest, and F. J. E. Comeau, "Using long short term memory and convolutional neural networks for driver drowsiness detection," *Accident Anal. Prevention*, vol. 156, Jun. 2021, Art. no. 106107.

[3] M. Hashemi, A. Mirrashid, and A. B. Shirazi, "Driver safety development: Real-time driver drowsiness detection system based on convolutional neural network," *Social Netw. Comput. Sci.*, vol. 1, no. 5, p. 289, Sep. 2020, doi: 10.1007/s42979-020-00306-9.

[4] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.

[5] *RISC-V Specification*, vol. 1, RISC-V, Unprivileged Spec V, Cham, Switzerland, 2019.

[6] S. K. Moore, "RISC-V AI chips will be everywhere," IEEE Spectrum, New York, NY, USA, Tech. Rep., 2022.

[7] P. Dabbelt *et al.* (2020). *GNU Toolchain for RISC-V, Including GCC*. [Online]. Available: https://github.com/riscv/riscv-gnu-toolchain

[8] S. Arif, M. Arif, S. Munawar, Y. Ayaz, M. J. Khan, and N. Naseer, "EEG spectral comparison between occipital and prefrontal cortices for early detection of driver drowsiness," in *Proc. Int. Conf. Artif. Intell. Mechatronics Syst. (AIMS)*, Apr. 2021, pp. 1–6.

[9] S. Murugan, J. Selvaraj, and A. Sahayadhas, "Detection and analysis: Driver state with electrocardiogram (ECG)," *Phys. Eng. Sci. Med.*, vol. 43, no. 2, pp. 525–537, Jun. 2020.

[10] A. T. Satti, J. Kim, E. Yi, H.-Y. Cho, and S. Cho, "Microneedle array electrode-based wearable EMG system for detection of driver drowsiness through steering wheel grip," *Sensors*, vol. 21, no. 15, p. 5091, Jul. 2021.

[11] G. Sikander and S. Anwar, "Driver fatigue detection systems: A review," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 6, pp. 2339–2352, Jun. 2018.

[12] V. S. Kumar, S. N. Ashish, I. V. Gowtham, S. P. A. Balaji, and E. Prabhu, "Smart driver assistance system using raspberry pi and sensor networks," *Microprocessors Microsyst.*, vol. 79, Nov. 2020, Art. no. 103275.

[13] S. Gupta, P. Jain, and E. Rufus, "Drowsy driver alerting system," in *Proc. 2nd Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA)*, Mar. 2018, pp. 1665–1670.

[14] A. A. Suhaiman, Z. May, and N. A. Rahman, "Development of an intelligent drowsiness detection system for drivers using image processing technique," in *Proc. IEEE Student Conf. Res. Develop. (SCOReD)*, Sep. 2020, pp. 233–236.

[15] W. Deng and R. Wu, "Real-time driver-drowsiness detection system using facial features," *IEEE Access*, vol. 7, pp. 118727–118738, 2019.

[16] B. Reddy, Y.-H. Kim, S. Yun, C. Seo, and J. Jang, "Real-time driver drowsiness detection for embedded system using model compression of deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jul. 2017, pp. 121–128.

[17] B. K. Savas and Y. Becerikli, "Real time driver fatigue detection system based on multi-task ConNN," *IEEE Access*, vol. 8, pp. 12491–12498, 2020.

[18] G. Zhang, K. Zhao, B. Wu, Y. Sun, L. Sun, and F. Liang, "A RISC-V based hardware accelerator designed for Yolo object detection system," in *Proc. IEEE Int. Conf. Intell. Appl. Syst. Eng. (ICIASE)*, Apr. 2019, pp. 9–11.
[19] A. Gonzalez and C. Hong, "A chipyard comparison of NVDLA and Gemmini," Berkeley, CA, USA, Tech. Rep. EE 290-2, 2020.
[20] F. Farshchi, Q. Huang, and H. Yun, "Integrating NVIDIA deep learning accelerator (NVDLA) with RISC-V SoC on FireSim," in *Proc. 2nd Workshop Energy Efficient Mach. Learn. Cognit. Comput. Embedded Appl. (EMC)*, Feb. 2019, pp. 21–25.
[21] E. Gholizadehazari, T. Ayhan, and B. Ors, "An FPGA implementation of a RISC-V based SoC system for image processing applications," in *Proc. 29th Signal Process. Commun. Appl. Conf. (SIU)*, Jun. 2021, pp. 1–4.
[22] S. Abtahi, M. Omidyeganeh, S. Shirmohammadi, and B. Hariri, "YawDD: A yawning detection dataset," in *Proc. 5th ACM Multimedia Syst. Conf. (MMSys)*, 2014, pp. 24–28.
[23] A. Franco, J. Charlong, and E. Badger. *A Python Library for Fractional Fixed-Point (Base 2) Arithmetic and Binary Manipulation With Numpy Compatibility*. Accessed: 2020. [Online]. Available: https://github.com/francof2a/fxpmath
[24] M. Perotti, P. D. Schiavone, G. Tagliavini, D. Rossi, T. Kurd, M. Hill, L. Yingying, and L. Benini, "HW/SW approaches for RISC-V code size reduction," in *Proc. Workshop Comput. Archit. Res. RISC-V (CARRV)*, 2020, pp. 1–8.
[25] LowRISC. *Ibex Core Documentation*. Accessed: 2017. [Online]. Available: https://ibex-core.readthedocs.io/en/latest/
[26] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020, doi: 10.1109/MM.2020.2996616.
[27] R. Herveille, "WISHBONE system-on-chip (SoC) interconnection architecture for portable IP cores," OpenCores, Amsterdam, The Netherlands, Tech. Rep., 2010.
[28] (2011). A R M Limited. AXI Spec. [Online]. Available: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI%4_specification.pdf
[29] MathWorks. *MATLAB Support Package for Raspberry PI Hardware*. Accessed: 2014. [Online]. Available: https://www.mathworks.com/hardware-support/raspberry-pi-MATLAB.html
[30] MathWorks. *Deep Learning Toolbox Converter for Tensorflow Models*. Accessed: 2017. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/64649-deep-learning-toolbox-converter-for-tensorflow-models
[31] Digilent. *Nexys 4 DDR Board*. Accessed: 2013. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-4-ddr/start

SEYED KIAN MOUSAVIKIA was born in Urmia, Iran. He received the B.Sc. degree (Hons.) in electronics engineering from the University of Urmia, in 2019, where he is currently pursuing the M.Sc. degree in electronics engineering. From 2018 to 2019, he was a Researcher at Urmia University, where he was involved in the design of a signal generator based on direct digital synthesis method (DDS) on field programmable gate arrays. In 2020, he was a Researcher for the project between TUBİTAK (The Scientific and Technological Research Institution of Turkey) and the Ministry of Science, Research and Technology, Iran, developing RISC-V based on SoC system for driver fatigue detection algorithms. His current research interest includes HW/SW codesign with the emphasis on deep learning and machine learning algorithms on field programmable gate arrays.

ERFAN GHOLIZADEHAZARI was born in Urmia, Iran. He received the B.Sc. and M.Sc. degrees (Hons.) in electronics from Istanbul Technical University, Turkey, in 2021. From 2017 to 2018, he was a Researcher at Urmia University, Urmia, where he was involved in the IR signal decoding project using NEC protocol for controlling projection systems. His undergraduate research was about wireless charger designing and implementation. From 2020 to 2021, he was a Graduate Researcher for the project between TUBİTAK (The Scientific and Technological Research Institution of Turkey) and Ministry of Science, Research and Technology, Iran, developing RISC-V based on SoC system for driver fatigue detection algorithms. His research interests include microprocessor architecture, embedded systems, and HW/SW codesign.

MORTEZA MOUSAZADEH was born in Urmia, Iran. He received the B.S. degree in electrical engineering from the Iran University of Science and Technology, Tehran, Iran, in 2003, and the M.S. degree in electrical engineering and the Ph.D. degree in microelectronics from Urmia University, Urmia, in 2006 and 2014, respectively. He is currently a Professor at Urmia University. His research interests include mixed mode IC design, data converter, and AI accelerator.

SIDDIKA BERNA ORS YALCIN received the bachelor's and M.Sc. degrees in electronics and communication engineering from Istanbul Technical University (ITU), Turkey, in 1995 and 1998, respectively, and the Electrical Engineering degree in applied sciences from Katholieke Universiteit Leuven, Belgium, in 2005. She is currently an Associate Professor at ITU. Her main research interests include cryptography, embedded systems, and side-channel attacks.