IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 2 December 2024, accepted 7 January 2025, date of publication 16 January 2025, date of current version 23 January 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3530296

RESEARCH ARTICLE

Low-Cost Driver Monitoring System Using Deep Learning

HADY A. KHALIL[1,2], SHERIF A. HAMMAD[2], HOSSAM E. ABD EL MUNIM[3], AND SHADY A. MAGED[1]

[1]Department of Mechatronics Engineering, Faculty of Engineering, Ain Shams University, Cairo 11571, Egypt
[2]Garraio for Software Innovations, Nasr City, Cairo 11816, Egypt
[3]Department of Computer and Systems Engineering, Faculty of Engineering, Ain Shams University, Cairo 11571, Egypt

Corresponding author: Hady A. Khalil (ha@hadysk.com)

ABSTRACT Driver monitoring systems are becoming an essential part of Advanced Driver Assistance Systems (ADAS) safety features in modern vehicles. The U.S. National Highway Traffic Safety Administration reports that drowsy/fatigued driving results in almost 100,000 road accidents per year. Driver's fatigue can have different causes, such as lack of sleep, long journeys, restlessness, mental pressure and alcohol consumption. Early monitoring systems relied on data from vehicle sensors, and modern systems commonly use driver's eye tracking. Recently, there has been growing interest in utilizing machine vision and deep learning for driver monitoring. Using machine vision can create more advanced driver monitoring systems capable of detecting driver attention state as well as other features like smartphone usage while driving and seat belts. Machine vision systems usually require extensive processing power, which raises the cost of such systems. In this paper, we present a low-cost driver monitoring system using a $15 Raspberry Pi Zero 2 W board and deep learning CNN to deliver a system capable of monitoring and identifying different states of the driver like safe driving, distracted, drowsy, and smartphone usage, the system achieves an inference rate for 10 Frames Per Second (FPS) and above 90% accuracy with the testing dataset. In addition to the deep learning CNN which runs on Raspberry Pi CPU, we utilize the Raspberry Pi GPU to run a head pose estimation algorithm to boost the system's accuracy.

INDEX TERMS Deep learning, machine learning, AI, Raspberry Pi, driver monitoring system, tinyML, YOLO, OpenCL, CNN, embedded systems.

I. INTRODUCTION

A. BACKGROUND AND MOTIVATION

According to a survey conducted by the Centers for Disease Control and Prevention (CDC), an estimated 1 in 25 adults reported having fallen asleep while driving in the last 30 days. It was found that drivers who sleep 6 hours or less are more likely to report falling asleep while driving than drivers who sleep 7 or more hours [1].

The National Highway Traffic Safety Administration (NHTSA) estimated in 2017 that 91,000 police-reported crashes involved drowsy drivers. These crashes led to an estimated 50,000 people injured and nearly 800 deaths. In 2021 NHTSA reported the number of vehicle fatalities involving drowsy drivers was 684 or 1.6% of total fatalities in 2021, This represents 8.2% increase from 632 in 2020. The number of fatalities in distraction-affected crashes, i.e., a crash involving at least one driver who was distracted, was 3,522, or 8% of total fatalities in 2021. This represents a 12% increase from 3,154 in 2020 [2].

The rise in the number of distraction-related accidents can be attributed to the increased automation in modern vehicles. Almost all modern vehicles come with Level 2 automation, according to the research by Canalys [3] 3.5 million cars sold worldwide had level 2 autonomy driving features in Q4 2020, growing by 91% year-on-year. Modern vehicles offer a suite of driver safety and assistance features, These technologies are known as Advanced Driving Assistance Systems (ADAS). Two of the main features of any modern ADAS are Lane Keep Assist (LKA) and Adaptive Cruise

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

Control (ACC). Lane Keep Assist (LKA) is a safety feature designed to help drivers stay within their lane. If the system detects that the car is moving out of its lane without the use of turn signals, it will either alert the driver or make small steering adjustments to keep the car in the correct lane.

Adaptive Cruise Control (ACC) is an advanced version of regular cruise control. With regular cruise control, the driver sets the vehicle to a certain speed, and the cruise control system maintains that speed regardless of the traffic ahead of the vehicle. This system requires 100% of the driver's attention. In contrast, adaptive cruise control can adjust the speed dynamically based on other vehicles in traffic. For example, if the adaptive cruise control is set to 70 MPH and the vehicle in front slows down to 50 MPH, the system will slow down the vehicle from 70 MPH to 50 MPH while maintaining a safe distance from the vehicle in front.

By combining LKA and ACC, the vehicle can almost drive autonomously when traveling on highways, which typically covers long distances without any traffic lights, crossroads, or sharp turns. Therefore, a vehicle with LKA and ACC can continue driving for hours with minimal driver intervention. As a result of this technology, drivers tend to over-trust ADAS technology and may engage in secondary activities, such as using a smartphone, eating, or talking to other passengers, without paying attention to the road [4]. According to the results of a survey conducted by Jennes et al. [5], even though vehicle owner's manual typically lists warnings and limitations regarding Adaptive Cruise Control (ACC), 72 percent of ACC owners in the survey said that they were not aware of any manufacturer's warnings or limitations about their ACC system.

As ADAS becomes more advanced, reaching higher levels of autonomy, the importance of in-cabin driver monitoring systems as part of ADAS has become clear. Therefore, governing bodies worldwide are starting to add regulations mandating that car manufacturers include driver monitoring systems in their vehicles [6].

In the EU, all motor vehicles of categories M and N are required to be equipped with Driver Drowsiness and Attention Warning (DDAW) systems. This regulation applies to all new vehicles from July 7, 2024. In the US, the SAFE (Stay Aware for Everyone) Act of 2021 bill was introduced, requiring the Department of Transportation (DOT) to conduct research regarding the installation and use of driver monitoring systems to minimize or eliminate driver distraction, driver disengagement, automation complacency, and the foreseeable misuse of advanced driver-assistance systems (ADAS) [7].

With these changes, many Automotive OEMs have developed their own implementation of driver monitoring systems (DMS). Most driver monitoring systems are based on eye tracking and head pose estimation. By using eye tracking, the system can estimate the gaze direction and determine whether the driver is looking straight ahead at the road or is being distracted and not focusing on the road [8].

B. LITERATURE REVIEW

Researchers have commonly used driver monitoring systems that rely on physical features and pose estimation. Swapnil et al. [9] developed a method that relies on eye detection to determine if a driver's eyes are open or closed as an indication of a drowsy state using EAR (Eye Aspect Ratio). EAR is a scalar value that responds to the opening and closing of the eyes by calculating the distance between vertical eye landmarks and horizontal eye landmarks, using this method the system is able to detect if the eyes are open or closed and it can detect blinking. The system also has facial detection for driver authentication, and it can send alerts by email or SMS if the driver's drowsy state is detected. The system minimum requirements are an Intel i3 processor and 4GB of RAM.

A more complex approach is the one developed by the authors in [10] which relies on an Xbox Kinect's camera and its ability for skeleton pose tracking. The system can detect drowsiness by head pose estimation based on head joint tracking, and it can also detect smartphone usage while driving based on tracking of left-hand distance relative to the head joint. The system relies on the windows developer toolkit for Kinect.

With the advancement in CNNs (Convolutional Neural Networks), AI on the Edge, and AI on the Edge devices, researchers have developed methods that utilize CNNs trained on eye tracking and yawning detection with large diverse datasets covering different scenarios like Yawning Detection Dataset (YawDD) and Driving Monitoring Dataset (DMD). YawDD is a dataset of videos of drivers recorded by an in-car camera. It is used by researchers to primarily develop and test algorithms and models for yawning detection, but also object recognition and tracking of face and mouth. Yawning is primarily used as an indicator of a driver's fatigue state. Similar to YawDD, DMD is another popular dataset of videos featuring drivers in a car. It is used by researchers to primarily develop and test algorithms and models for driver monitoring including smartphone usage and distracted driving [11].

Savaş et al. [12] proposed a multi-task CNN model using Dlib's facial landmarks algorithm to accurately identify the driver's eye and mouth information. Then, the system is trained with multi-task CNN models to determine driver fatigue state. The proposed model achieves 98% accuracy with YawDD.

Vural et al. [13] developed a method that relies on facial expressions to determine a driver's fatigue state instead of using eye-tracking methods. One of the most significant results from this research is that yawning was found to be a negative predictor of the 60-second window prior to a crash. Their testing revealed that, in the moments before falling asleep, drivers yawn less frequently rather than more often.

The authors in [14] proposed an algorithm that makes use of features learned using a convolutional neural network to explicitly capture various latent facial features and complex

non-linear feature interactions. the system achieved 92.33% validation accuracy in the dataset used.

Zhang et al. [15] used deep learning to detect the face area and nose location then this is used with a nose tracking algorithm and a neural network for yawning detection based on the extracted features such as nose tracking confidence degree, gradient features around edges of mouth, and face motion features. The proposed system achieved 92% accuracy on YawDD dataset.

Authors in [16] developed a fatigue detection algorithm based on CNNs. They built a multi-task cascading convolution neural network, combining 3 sub-networks. The network can be used for face detection and key point detection, including five key points: the left and right mouth corners, the center of the nose, and the centers of the left and right eyes, then a fatigue judgment model determines if the driver is in a fatigued state or not. The algorithm achieved 98.42% detection accuracy on CEW eye detection dataset and achieved 97.93% detection accuracy on YawDD dataset for yawning detection. The model was running on an Intel(R) Core(TM) i7-6700U, 8GB of memory, and NVIDIA's GTX 1080 Ti GPU.

Zhang et al [17] presented a novel approach of using a large language model (LLM) called Distracted Driving Language Model (DDLM). The model is designed to do complete body pose estimation including head and hands to detect distracted driving cases. The model was trained on the 100-Driver dataset and achieved 89% accuracy. When compared to other popular CNN models, SqueezeNet achieved an accuracy of 82% on the same data. The model inference was done locally on a system with 8GB of GPU's VRAM.

A different approach is to utilize multiple cameras with different view points similar to the system proposed by Chen et al. [18]. The system uses a multi-modal and multi-view fusion FER model that can accurately recognize facial expressions regardless of lighting conditions or head poses, by utilizing 3 different image sources: RGB camera, IR camera, and depth camera. The multi-modal and multi-view fusion approach achieves an accuracy of over 95% when recognizing drivers' facial expressions in real-world scenarios, even in poor lighting conditions and different head poses.

Many of the methods presented in the literature that utilize deep learning and CNNs tend to use powerful CPUs and GPUs. This hardware has high cost and high power consumption which has led to increasing interest in utilizing edge devices and AI on the edge to develop systems that run on more cost-efficient and power-efficient hardware.

C. AI ON THE EDGE DEVICES

AI on the edge is the method of decentralizing the machine learning computation from using a centralized powerful computer to smaller distributed computing devices that are closer to the data source, where data acquisitions, processing and decision are made on these devices, hence the name edge devices as in they are distributed on the edge of the system close to the data source. In machine vision, edge computing devices must have the ability to obtain images through a camera, connect to the internet for IoT applications, and process data quickly enough to support real-time applications. Several devices have been developed targeting edge computing and are actively used in different machine vision applications, Table 1 shows different edge devices and applications.

TABLE 1. Edge devices and applications.

Paper	Edge Device	Application
[19]	Raspberry Pi 4 + NM500	Face Recognition
[20]	Raspberry Pi 3	Smart Surveillance
[21]	Raspberry Pi 3	Air Pollution Detection System
[22]	TI-TDA4VM	Driver Monitoring System

Raspberry Pi boards are commonly used in Edge AI applications [23], but other HW options are also available. for example, Google's Coral USB accelerator, NVIDIA's Jetson TX2, and specialized AI boards like TI-TDA4VM.

TI-TDA4VM is used by Hariharan et al. [22] to build a driver monitoring system to detect head pose estimation as well as yawn and eye detection to determine whether the driver is distracted, drowsy, or alert.

D. EDGE AI MODELS

Edge AI devices tend to have limited hardware resources, to deal with this constraint, deep learning models can be designed and optimized to fit on limited hardware. For example: MobileNet CNN was designed with the aim of running on smartphones, MobileNet uses depth-wise convolution as opposed to normal convolution, with this design MobileNet can achieve same accuracy as GoogleNet and VGG 16 but with a smaller number of parameters [24].

Model optimization techniques can also be used to reduce the amount of resources required for the CNN model. An example of these techniques is Model Quantization. Machine learning models typically use floating-point numerical representation to achieve high-accuracy calculations, but floating-point operations are computationally expensive. By using quantization, a machine learning model is converted to use integers instead of floating point, which significantly decreases the size of the model and its computational requirement as shown in Fig. 1.

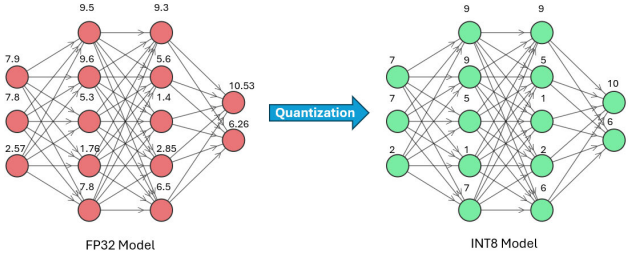

FIGURE 1. Machine learning model quantization.

Quantization can be done during training or post-training. Quantization during training is called quantization aware training, it involves training a model with quantized parameters which results in a higher accuracy quantized model compared to post-training quantization. Jacob et al. [25] proposed a method that simulated quantization effects in the forward pass of training, while backpropagation still happens with all weights and biases stored in floating-point representation. Using this method with ImageNet results in improved performance, with only a 2% lower accuracy in the quantized model compared to the unquantized model.

In post-training optimization, the model is trained with the maximum floating-pointing precision possible resulting in a high accuracy model at the end of training. After the training is complete, the weights of the model can be converted to types with reduced precision such as 16-bit float or 8-bit integers. Post-training quantization provides more flexibility as the training is completed once, then varying quantization techniques can be used to provide the best performance to accuracy ratio, also post-training quantization allows the generation of different quantized models from the main model to be deployed on different devices based on the hardware requirements and capabilities without having to go through the training process again. Rokh et al. presents a comprehensive overview of different quantization methods [26].

E. PROBLEM DEFINITION

Driver monitoring systems that rely on machine vision and CNNs are usually resource intensive, requiring powerful CPUs and GPUs [16], [17]. The need for powerful hardware in machine vision applications results in systems that have high costs and high power consumption. This has led researchers to start developing systems that are less resource intensive requiring less power and cheaper hardware to run machine vision applications.

One of the proposed systems to solve the resources requirement issue for driver monitoring systems is developed by Kim et al. [23], The system uses deep learning CNNs (Multi-Task Mobilenets) to build a lightweight driver monitoring system and by utilizing a resource-sharing mechanism between two devices, main block of the system is implemented on in-vehicle Raspberry Pi SBC and sub-device blocks are implemented on a resource sharing device such as the driver's mobile device. The proposed system was able to achieve twice as much FPS compared to using a single Raspberry Pi (from 2.15FPS to 4.47FPS) and showed 11.0% higher accuracy.

Alajlan and Ibrahim [27] developed and trained multiple tinyML CNN models for driver monitoring that can run on low-power embedded systems like Raspberry Pi or STM microcontrollers. The developed deep learning models are MobileNet-V2, SqueezeNet, AlexNet, and MobileNet-V3, achieving accuracies of 0.9960, 0.9947, 0.9911, and 0.9832, respectively, in identifying driver drowsiness status (i.e., yawning, non-yawning, closed eyes, and open eyes).

F. DRIVER MONITORING SYSTEMS USING RASPBERRY PI AS EDGE DEVICE

Similar to Kim et al. [23], other researchers have used the Raspberry Pi in driver monitoring systems. Chellappa et al. [28] proposed a method using Raspberry Pi 3 where a Haar cascade classifier was applied to detect the blink duration of the driver, and the eye aspect ratio (EAR) was computed using the Euclidean distance between the eyes, detecting the eye-closing rate every 0.5 seconds. A similar method presented by Wong and Lau [29] also uses a Raspberry Pi 3 and relies on facial landmarks, eye detection, and PERCLOS (percentage of eyelid closure) ratio to determine the driver's drowsy state.

Hossain and George [30] proposed a driver monitoring system that also relies on the eye aspect ratio method (EAR) and takes advantage of the IoT capabilities of the Raspberry Pi 3 in that it can send an email alert if the driver is in a drowsy state.

All the previous methods mentioned only use the Raspberry Pi 3 CPU and mostly rely on eye-aspect ratio detection. However, the Raspberry Pi GPU often goes underutilized in the literature. The Raspberry Pi 3 (and the Raspberry Pi Zero 2 W) have a GPU called VideoCore IV capable of a theoretical 24 GFLOPS of processing power.

G. OUR CONTRIBUTION

In this paper, a low-cost driver monitoring system is presented. Using YOLOv8 deep learning CNN running on a $15 Raspberry Pi Zero 2 W, the system can detect if a driver is distracted, drowsy, or using a smartphone while driving. In parallel to running YOLOv8 CNN on the Raspberry Pi Zero 2 W CPU, a head pose estimation algorithm was developed to run on the Raspberry Pi GPU using OpenCL to boost the accuracy of the driver monitoring system. To the best of our knowledge, no prior research has utilized the Raspberry Pi GPU for driver monitoring systems. The system achieves a 10 FPS inference rate and over 90% accuracy, offering a high inference rate at a low cost compared to similar research in the literature. In addition to the developed system, a diverse dataset was created from different distributions suitable for the challenging environments of a driver monitoring system.

The rest of the paper is presented as follows: Section II provides an overview of the methods used, hardware choice, and the proposed algorithm. Real world experimental results are presented in Section III. In Section IV, the results and system performance are discussed. Finally, conclusions are presented in Section V.

II. MATERIALS AND METHODS

A. RASPBERRY PI ZERO 2 W

Building a machine vision application requires careful consideration of various hardware constraints to ensure optimal performance. The most critical factor in selecting the hardware for machine vision is memory (specifically

TABLE 2. Comparison of Low-Cost SBCs.

Device	Processor	Memory	GPU	Price
Raspberry Pi Zero 2 W	BCM2710A1 Quad-Core Cortex-A53 1GHz	512MB	VideoCoreIV	$15
Raspberry Pi 3 Model B+	BCM2837B0 Quad-Core Cortex-A53 1.2 GHz	1GB	VideoCoreIV	$35
Banana Pi BPI-M2 Zero	Allwinner H2+ Quad-core Cortex-A7 1.2 GHz	512MB	Mali-400 MP2	$18
Orange Pi Zero3	Allwinner H618 Quad-Core Cortex-A53 1.5 GHz	1GB - 4GB	Mali-G31 MP2	$23.99 for the 1GB model
Raspberry Pi 4 Model B	BCM2711 Quad-Core Cortex-A72 1.5 GHz	1GB - 8GB	VideoCore IV	$30.67 for the 1GB model

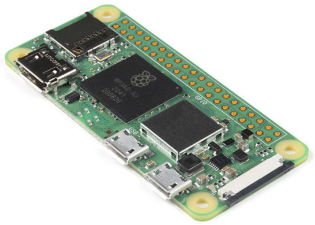

FIGURE 2. Raspberry Pi Zero 2 W SBC.

RAM), Convolution Neural Networks (CNNs) contain large amounts of parameters often in the range of millions. With large enough memory any CNN model can be used regardless of its number of parameters, and then the next limitation is the processing power. The speed of the processor will determine how long one prediction takes (inference time). The third constraint is the camera, the camera should provide adequate quality that fits the application needs, as well as having a communication interface suitable to be connected to the processing hardware.

The Raspberry Pi Zero 2 W (Fig. 2) is a low-cost ($15) single-board computer (SBC) with the same quad-core 64-bit ARM Cortex-A53 processor as the Raspberry Pi 3, only clocked lower at 1 GHz down from 1.4 GHz, and with 512 MB of RAM instead of 1 GB, it has a VideoCore IV GPU, and a dedicated CSI-2 camera connector. The Raspberry Pi Zero 2 W also supports 32-bit and 64-bit operating systems, making it a good fit for machine learning development and testing.

The Raspberry Pi Zero 2 W is one of the most popular single-board computers in Internet of Things (IoT), machine

TABLE 3. Raspberry Pi Zero 2 W specifications.

Device Name	VideoCoreIV GPU	ARM Cortex-A53 CPU
Processing Cores	12 QPUs	4 Cores
Officially Supported Graphics APIs	OpenGL-ES 1.1/2.0	N/A
Unofficially Supported Graphics APIs	OpenCL 1.2/Vulkan	OpenCL 1.2
Clock Frequency	250 MHz	1 GHz
Memory	512MB shared with the CPU	512MB shared with the GPU

learning, and embedded applications. there are many other low-cost devices on the market with each presenting different set of features compared to the Raspberry Pi Zero 2 W, some of the Raspberry Pi competitors are presented in Table 2.

We chose the Raspberry Pi Zero 2 W instead of other Raspberry Pis presented in Table 2 as it offer similar hardware only with the CPU running at a lower clock and with smaller amount of RAM, and compared to the hardware offered by Banana Pi and Orange Pi, the Raspberry Pi has better software support due to its wider adoption in industrial and hobbyist applications.

For the camera setup, the Raspberry Pi Zero 2 W offers multiple options as it has a CSI-2 camera connector as well as a micro-USB 2.0 port, enabling compatibility with a wide range of cameras. In our case, we opted for a 5-megapixel Raspberry Pi night vision camera as shown in Fig. 3.

FIGURE 3. Raspberry Pi IR camera.

Using an active IR camera is essential for the driver monitoring system as regular camera's quality is affected by lighting conditions which in turn will affect the CNN prediction accuracy. IR cameras eliminate this issue as they provide a clear picture in low light conditions as in the case of driving at nighttime.

B. RASPBERRY PI ZERO 2 W GPU

The Raspberry Pi Zero 2 W is equipped with a quad-core 64-bit ARM Cortex-A53 processor and VideoCore IV GPU; by utilizing both the CPU and GPU, the processing load can be shared between them, leading to improved performance and reduced execution time. The VideoCoreIV GPU is capable of a theoretical 24 GFLOPS of performance and support

multiple graphics and computation APIs. Specification of the Raspberry Pi Zero 2 W CPU and GPU are presented in Table 3.

Daniel Stadelmann developed VC4CL [31], which is an implementation of the OpenCL 1.2 standard for the VideoCore IV GPU (found in Raspberry Pi 1, 2, 3, Zero and Zero 2 W models).

OpenCL is an open standard for parallel computing developed by the Khronos group.It enables software developers to write code that can run across different types of hardware, such as CPUs, GPUs, DSPs (Digital Signal Processors), and FPGAs (Field Programmable Gate Arrays). Its main purpose is to take advantage of the processing power of these devices to accelerate performance in computationally intensive tasks like scientific computing, image processing, and machine learning. OpenCL supports C and C++ programming making it easy to use with machine learning applications and computation as shown in Fig. 4.

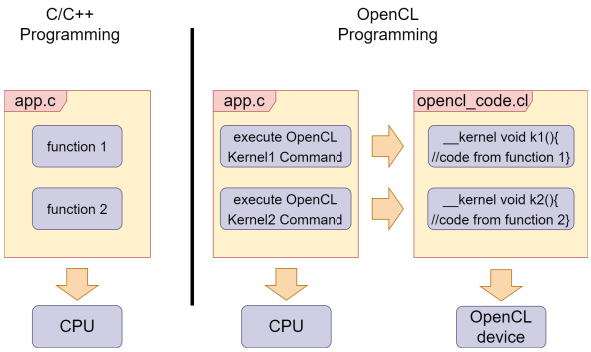

FIGURE 4. C/C++ programming VS OpenCL programming.

We utilized OpenCL and VC4CL to run head pose estimation algorithm on the Raspberry Pi GPU described in section II-D. This way, the CPU can run the CNN inference, while the GPU runs the head pose estimation algorithm. The GPU is not used to run the CNN model, as the CPU is needed to prepare the data and execute the GPU kernel, and due to the large number of parameters in the CNN model and limited memory bandwidth of the Raspberry Pi Zero 2 W, it is faster to run the CNN model on the CPU instead of the GPU.

C. YOLOv8

You Only Look Once (YOLO) is an object detection CNN introduced in 2016 by Redmond et al. [32]. YOLO is a single-shot detector that can directly detect and classify multiple objects in a single pass, It has evolved over the years until the most recent version, YOLOv8.

YOLOv8 provides high accuracy while maintaining a small model size. It has five different model designs with increasing size and accuracy. The smallest YOLOv8 model is YOLOv8n, which has 3.2 million parameters and 37.3 mAP 50-95 accuracy on the COCO dataset.

YOLOv8 architecture is divided into two main parts: a backbone and a head (Fig. 5). The backbone layer consists of a series of convolution layers for feature extraction; then,

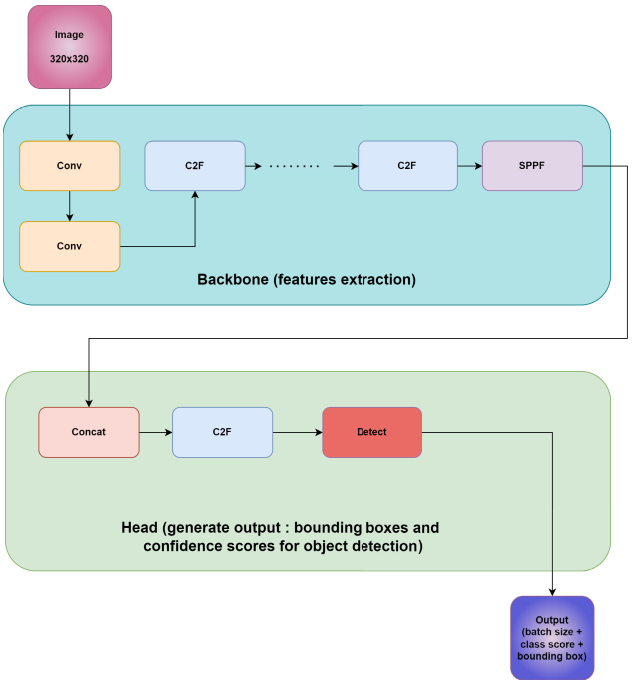

FIGURE 5. YOLOv8 architecture.

the features are passed to the head for object detection. Terven et al. provide a comprehensive review of the architecture of different YOLO versions [33].

YOLOv8 framework provided by Ultralytics supports generating different quantized versions of the YOLOv8 model, we used the framework to generate an INT8 quantized version of YOLOv8n model to test its accuracy and performance as shown in Table 4. For our driver monitoring system, a modified version of YOLOv8n CNN was used, replacing the regular convolution layers with GhostNet Modules [34]. The GhostNet module divides the initial convolutional layer into two segments, employing a reduced number of filters to create multiple feature maps. Subsequently, cheap transformation operations are applied to produce ghost feature maps efficiently. GhostNet convolution reduce the computational requirement of the CNN while maintaining the same level of accuracy. By utilizing YOLOv8n with GhostNet as well as reducing the number of layers in YOLOv8n, the performance of the neural network is improved from 1.4 FPS to 3.0 FPS. Compared to other CNNs, YOLOv8 offers the best performance and accuracy combination as shown in Table 4, it is also supported by all the major machine learning frameworks like PyTorch, TensorFlow, and NCNN.

We trained YoloV8 model to be able to detect 4 different classes (driver states) with the input being a 320 × 320x3 image captured from a camera, the 4 classes are as follows:

1) Safe Driving: Driver looking ahead with eyes on the road.
2) Distracted: Driver looking away from the road.
3) Drowsy: Driver is tired (yawning, sleeping, or closed eyes).
4) Smartphone: Driver using a smartphone while driving.

TABLE 4. Different CNN models and their performance.

CNN Model	Description	FPS	Accuracy (mAP50 for COCO dataset)
NanoDet	NanoDet is a lightweight anchor-free object detection model.	13.0	20.6%
PP-PicoDet	PP-PicoDet is a lightweight anchor-free object detection model offering improved performance over similar models like NanoDet	7.5	27.0%
YOLOv5n	The smallest YOLOv5 model available, it is trained and stored with 32-bit floating-point precision (float32).	1.6	22.5%
YOLOv8n	The smallest YOLOv8 model available, it is trained and stored with 32-bit floating-point precision (float32)	1.4	37.3%
YOLOv8n int8 quantized	YOLOv8n model quantized to int8, int8 quantization improves performance at the cost of reduced accuracy	3.1	32.0%
YOLOv8n with GhostNet	YOLOv8n model with GhostNet convolution layers to improve performance while maintaining high accuracy compared to the original model.	3.0	36.5%

D. HEAD POSE ESTIMATION

The developed head pose estimation algorithm relies on dlib's face landmark detector. Dlib is a C++ library containing machine learning algorithms and tools, it provides a trained face landmark detector that uses cascaded regression trees to estimate the facial landmarks position with high accuracy [35].

Dlib face landmarks detector is commonly used in driver monitoring system that rely on eye aspect ratio (EAR) or head pose estimation. Mohanty et al. [36] use dlib face landmarks to detect Eye Aspect Ratio (EAR) to monitor the driver's blinking pattern and Mouth Aspect Ratio (MAR) based on calculating the Euclidean distance from eyes and mouth landmarks. The proposed method by [37] combines the face landmarks detected by dlib with a CNN to determine the driver's drowsy state.

Pondit et al. [38] take the face landmarks from dlib and calculate the eye aspect ratio and mouth aspect ratio similar to previous methods, but add head pose estimation by using OpenCV's perspective-n-point algorithm (solvePnP) [39] to detect the driver's head pose to determine if the driver is looking ahead, to the left, or to the right.

Our method uses appearance-based head pose estimation algorithm in order to utilize the Raspberry Pi GPU and OpenCL. Facial landmarks are detected using dlib's facial landmarks detector and then the landmarks are passed to head pose estimation algorithm. The algorithm calculates a center gaze point for the head from the landmarks and then compares it to a ground-truth dataset as shown in Fig. 6.

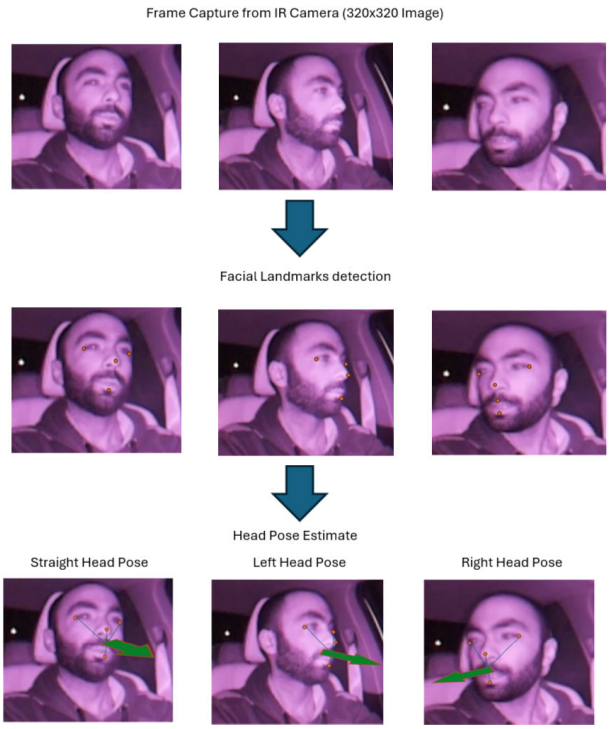

FIGURE 6. Head pose estimation.

The algorithm detects distracted state if the head pose estimate detected is more than 45 degrees orientation from normal head orientation looking straight at the road. Utilizing the GPU parallelism ability, we can find out the head pose from the ground-truth dataset while in parallel running YOLOv8 CNN on the CPU.

Both the head pose estimation algorithm and YOLOv8 model contribute to the accuracy of driver's state detection, in our algorithm, The head pose estimation contributes by 50% to the accuracy of the system, while in YOLOv8 the confidence threshold is set at 40%, so model predictions below 40% are ignored. Between 40% and 70% prediction accuracy, YOLOv8 contributes to the accuracy of the system by 50% and above 70% prediction accuracy, we increase YoloV8 contribution to the system's accuracy to 60% and the head pose estimation is lowered to 40%. We increase the contribution of YOLOv8 to the accuracy of the system for predictions above 70% because during our testing we noticed that YOLOv8 gives consistent positive predictions at 70% confidence. The complete driver monitoring algorithm is described in algorithm 1.

E. DATASET

1) DATASET CHOICE IN RELATION TO CAMERA POSITION

When building a dataset for a machine vision driver monitoring system, there are two main criteria to take into consideration. The first is finding the optimal camera position

Algorithm 1 Driver Monitoring Algorithm

Require: calibrated camera matrix K, model points W
Capture image frame of the camera *Im*
Detect face landmark points using dlib face detector *ImgP*

if *faceDetected* == *true* **then**
Run OpenCL kernel to calculate head pose estimate on the Pi GPU
Run inference with YOLOv8 on captured image frame *Im*
Head pose estimate algorithm:
for $i <$ *ImgP.number* **do**
hpX +=ImgP[i,j]
end for
for $j <$ *ImgP.number* **do**
hpY +=ImgP[i,j]
end for
$hpX = \frac{hpX}{ImgP.number}$
$hpY = \frac{hpY}{ImgP.number}$
for $i <$ *PoseDataSet* **do**
poseErrX = hpX - PoseDataSet[i,j]
poseErrY = hpY - PoseDataSet[i,j]
if *poseErrX* $<$ *errTh* **then**
currPoseX = PoseDataSet[i,j]
end if
if *poseErrY* $<$ *errTh* **then**
currPoseY = PoseDataSet[i,j]
end if
end for
Read back head pose estimate (currPoseX,CurrPoseY) from the Raspberry Pi GPU
else
Run inference with YOLOv8 on captured image frame *Im*
end if
Estimate driver state based on YOLOv8 predication and head pose estimate
if *yoloP* > 0.4 && *yoloP* < 0.7 **then**
poseEstimate =0.5*yoloP + 0.5*poseNorm
else if *yoloP* > 0.7 **then**
poseEstimate =0.6*yoloP + 0.4*poseNorm
end if
if *poseEstimate* > 0.5 **then**
Report Driver State based on poseEstimate
end if

inside the vehicle. In Fig. 7 we show the two most common camera placement positions for driver monitoring systems.

Each camera position has its own advantages and disadvantages. In Table 5, we present a comparison between each camera position.

As demonstrated in Table 5, position 2 biggest advantage is that it has a direct view of the driver, unlike position 1, where the camera may be able to see the driver and passenger.

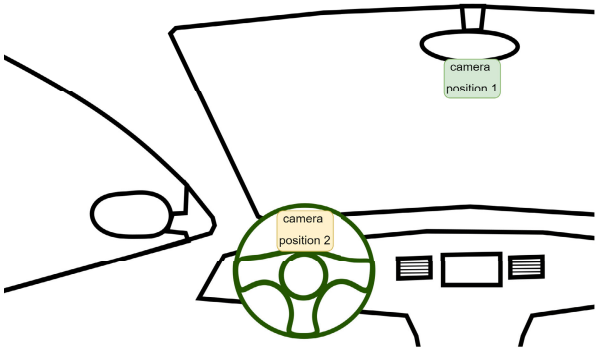

FIGURE 7. Driver monitoring system camera position.

TABLE 5. Comparison between driver monitoring camera positions.

	Camera Position 1	Camera Position 2
Direct view of the driver	No	Yes
Full view of the car cabin	Yes	No
Can be blocked during driving	No	Yes

Position 2 is ideal for systems that rely on eye tracking, while position 1 is ideal if the system is to be integrated as part of a dash cam recording system and it cannot be blocked by the driver hands during driving. In our case, we chose position 1 as it cannot be blocked while driving and it allows for future system updates to include other in-cabin monitoring applications like seat belt detection for drivers and passengers. Since the camera view is different between position 1 and position 2, the dataset needs to be chosen accordingly, If the dataset used to train the CNN model contained images taken from position 2, and the driver monitoring system camera was placed in position 1 during model deployment, the model's accuracy would be negatively affected.

2) DATASET CHALLENGES FOR DRIVER MONITORING SYSTEM

Once the camera position is defined, the second important criteria to take into consideration when building the dataset is that driver monitoring systems in real-world scenarios are subjected to images with high level of variation, this is due to different lighting conditions, different car designs, different drivers' body types and heights. Therefore, it's important that the dataset is from a wide range of different distributions covering as many conditions as possible.

Many of the driver monitoring systems datasets available online present a single distribution with images of similar lighting conditions and setup. In order to have a wide range of different distributions, we built a large dataset containing over 8,000 images by combining five driver monitoring

TABLE 6. Dataset distribution.

Classes	Number of Images
SafeDriving	2600
Distracted	2700
Drowsy	1300
SmartPhone	1900

TABLE 7. Training configurations.

Parameter	Value
Training Platform	Google Colab
GPU	NVIDIA V100 with 16GB of VRAM
System RAM	51GB
CUDA enabled	Yes
Epochs	500
Batch size	128
Optimization Algorithm	Stochastic Gradient Descent(SGD)
SGD Learning rate	0.01
SGD Momentum	0.9
Average Training Duration	3 Hours

datasets plus our own dataset of images. After applying data augmentation, the total dataset size increased to 22,000 images. The dataset includes images in different lighting conditions, multiple camera angles, and varying passenger body types. Our dataset can be found online on roboflow, which is the platform we used for data annotation and porting the dataset into Google Colab for model training [40].

The Dataset is divided into 4 classes (SafeDriving, Distracted, Drowsy and Smartphone) as shown in Table 6.

Fig. 8 shows an example from the dataset for each class.

Augmented versions of the images in the dataset are generated before training in order to add more diversity and variation in the dataset. We applied the following augmentations: horizontal flip, random crop, rotation, shear, gray scale, exposure increase and decrease, blur, and noise as shown in Fig. 9. The dataset was split into 70% training data, 15% validation data, and 15% testing data.

III. EXPERIMENTAL RESULTS

A. TRAINING AND TESTING ENVIRONMENT

Google Colab Pro was used for YOLOv8 model training, which gives access to NVIDIA V100 GPU with 16GB of VRAM and 51GB of System RAM [41]. YOLOv8 models and Colab projects can be found on YOLOv8's project GitHub [42]. The training configurations are listed in Table 7.

The Raspberry Pi Zero 2 W combined with an IR camera was used for model deployment and real-world testing. For the operating system, we chose Raspberry Pi OS 32-bit as the implementation of OpenCL (VC4CL) only supports 32-bit OS. Along with VC4CL for OpenCL, we used Tencent's NCNN machine learning framework to run YOLOv8 model.

TABLE 8. Model validation results.

Class	No. of Images	Accuracy(mAP50)
SafeDriving	750	95.9%
Distracted	750	96.8%
Drowsy	750	93.7%
Smartphone	750	75.0%

B. YOLOV8 TRAINING RESULTS AND EVALUATION

Training YOLOv8 model took around 3 hours, using 22GB of system RAM and 11GB of GPU VRAM. The loss graphs and confusion matrix are shown in Fig. 10 and Fig. 11.

The model should be able to generalize to a wide range of varying data and should not overfit or underfit to the dataset, while maintaining high accuracy.

In Fig. 10, the training graphs for bounding box (train/box_loss) and classification (train/cls_loss) show steady decrease and flattens towards the end, indicating that the model is improving accuracy over time, meanwhile the validation graphs for bounding box (val/box_loss) and classification (val/cls_loss) show gradual decrease with noise, indicating that the model is able to generalize to the validation data.

Our aim during the model training is to score above 90% accuracy for each class in order to have reliable predictions and avoid false positives. In Fig. 11, The model scores above 90% accuracy for the SafeDriving, Distracted, and Drowsy classes while only scoring below 90% accuracy for the Smartphone class. We observe that 35% percent of the background is misclassified as Distracted, the head pose estimation algorithm will help protect against this misclassification error. The Smartphone class is the hardest to detect with 81% accuracy and 56% of the background misclassified as Smartphone. Smartphones are challenging to detect because they come in different shapes and colors, can be covered by cases or the driver's hand, and have different positions while in use from one driver to another. The dataset samples for the Smartphone class needs to be increased to improve accuracy and avoid misclassifications.

C. MODEL VALIDATION AND REAL WORLD TESTING

The model validation results shown in Table 8 presents the accuracy and performance of the model.

The SafeDriving class had an mAP50 of above 95% suggesting that the model is able to reliably detect this class based on the validation data. Similar performance can be observed for the Distracted class and Drowsy class with an accuracy result of 96.8% and 93.7% respectively. The Smartphone class had the lowest mAP50 at 75.0%, indicating that the model struggles with identifying smartphone while driving. this can be attributed to several factors, it can be that

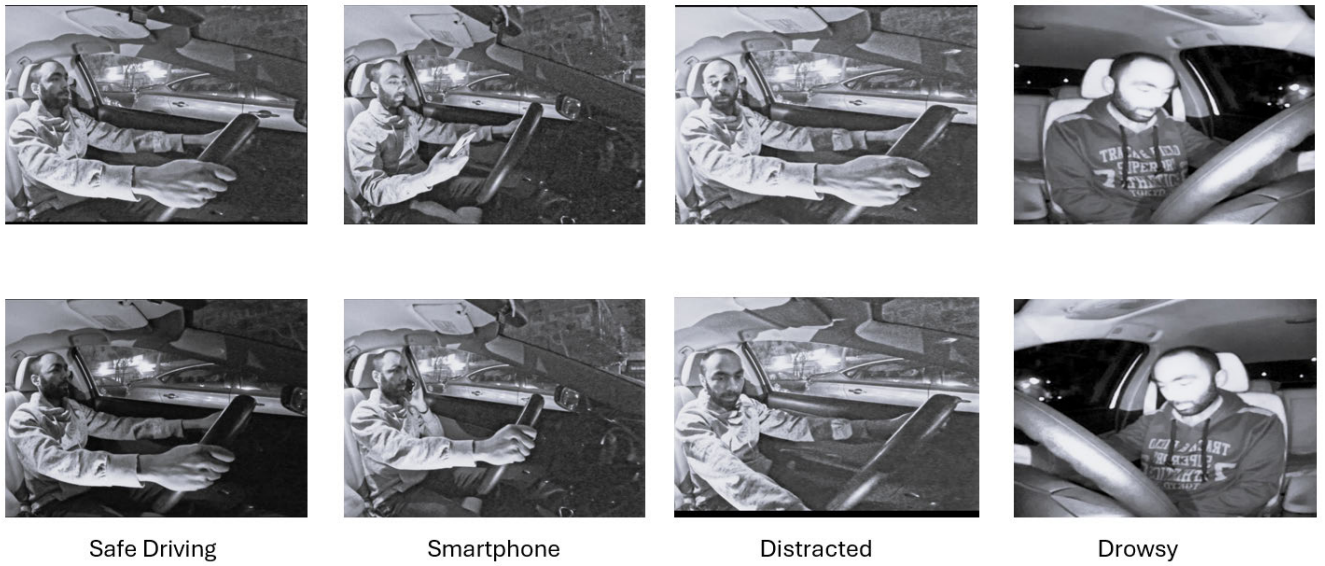

FIGURE 8. Dataset classes example.

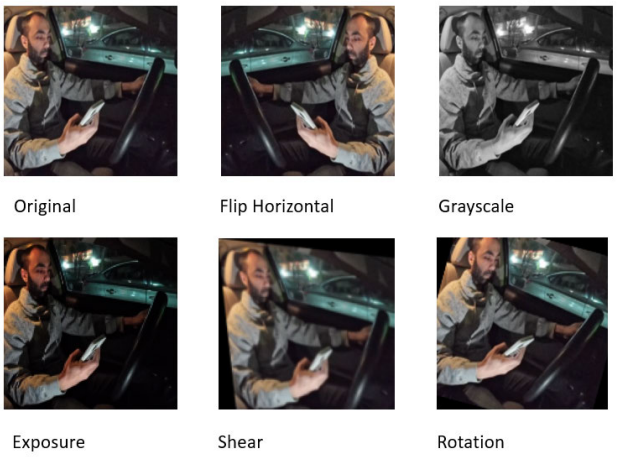

FIGURE 9. Dataset augmentation.

the dataset samples is not enough for the model to perform with high accuracy and more data is needed, it can also be attributed to the difficulty of identifying a smartphone inside a vehicle, in the testing data presented in Fig. 11 we saw that 56% of the background was misclassified as smartphone.

For real-world testing, we used data that the model had never seen before, either from our real-world testing scenarios captured in stationary vehicles or from other datasets. The model should be able to detect the driver's state with high accuracy and in parallel, the head pose estimation should boost the accuracy of the system when presented with new unseen data. Testing data are videos of drivers acting distracted, the driver would start to use a smartphone or look away from the road, and we monitor the performance of the model in real time. Presented in Fig.12 is the model performance on a sample of the real-world testing data. The sample shown was captured using our testing setup in a car at nighttime. The IR camera allows the system to work at nighttime without being affected by lighting condition as shown in Fig. 13.

The model was able to predict SafeDriving and Distracted state with accuracy of around 75% accuracy seen in the bounding box. The head pose estimation algorithm serves as another reference for the driver state and helps boost the overall accuracy of the system; during testing, the head pose estimation algorithm is able to detect SafeDriving and Distracted states(result of head pose estimate in blue text Fig. 12) consistently. The system determines the state of the driver by combining the two inputs from the YOLOv8 CNN model and the head pose estimation algorithm.

Drowsy images in the dataset were divided into images of drivers sleeping or yawning, As shown in Fig. 12, the model often confused the Drowsy and Distracted classes when the driver was sleeping. However, when presented with images of yawning drivers, the model was able to predict drowsy state with around 60% to 70% accuracy. This result suggests that images of sleeping drivers are better being added as part of the Distracted class instead of Drowsy, to avoid confusing the model. The model was able to detect the smartphone with around 50% accuracy. Smartphone detection has proven to be the most challenging class for the model, often confusing the background with a smartphone at 30% accuracy. Performance in smartphone detection indicates that the dataset samples for smartphone needs to be increased to improve accuracy. It is possible to use the distracted class to detect the driver looking away during texting and only use the smartphone class to detect smartphone usage for phone calls, that way the smartphone class will only contain data samples for phone calls and texting data samples can be used as part of the distracted class.

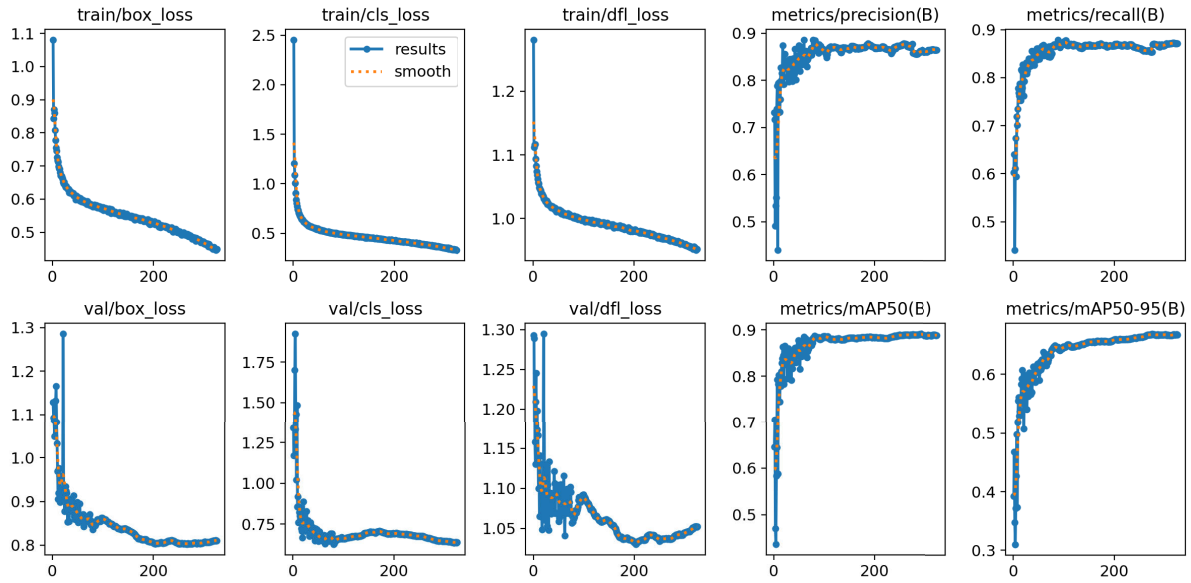

FIGURE 10. Training model loss graphs.

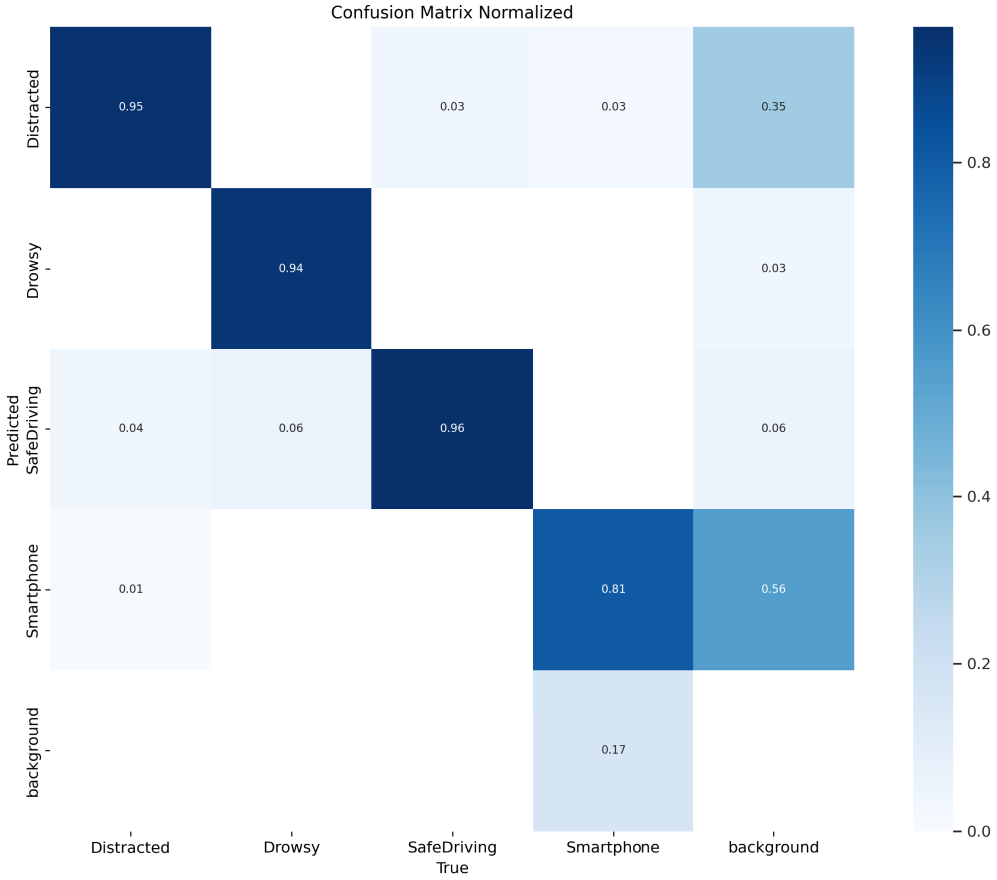

FIGURE 11. Trained model confusion matrix.

IV. DISCUSSION

Driver Monitoring Systems are becoming an essential part of Advanced Driver Assistance Systems (ADAS). In this paper the goal was to build a low-cost driver monitoring system using deep learning that is capable of detecting different driver states (SafeDriving, Distracted, Drowsiness,

TABLE 9. Comparison to related work.

Paper	Method	Detection Capabilities	Hardware	Detection Rate
[23]	CNN distributed over two devices to share hardware resources	Distraction + Drowsiness	Two Raspberry Pis	4.47 FPS
[28]	Eye aspect ratio (EAR)	Drowsiness	Raspberry Pi 3	2 FPS (0.5 second eye detection rate)
[29]	Facial landmarks-based head pose estimation and eye detection and PERCLOS (percentage eye closure)	Distraction + Drowsiness	Raspberry Pi 3	-
[30]	Eye aspect ratio (EAR) + Email alerts	Drowsiness	Raspberry Pi 3	-
Proposed System	CNN running on the Raspberry Pi CPU + head pose estimation running on the Raspberry Pi GPU	Distraction + Drowsiness + Smartphone	Raspberry Pi Zero 2 W	10 FPS

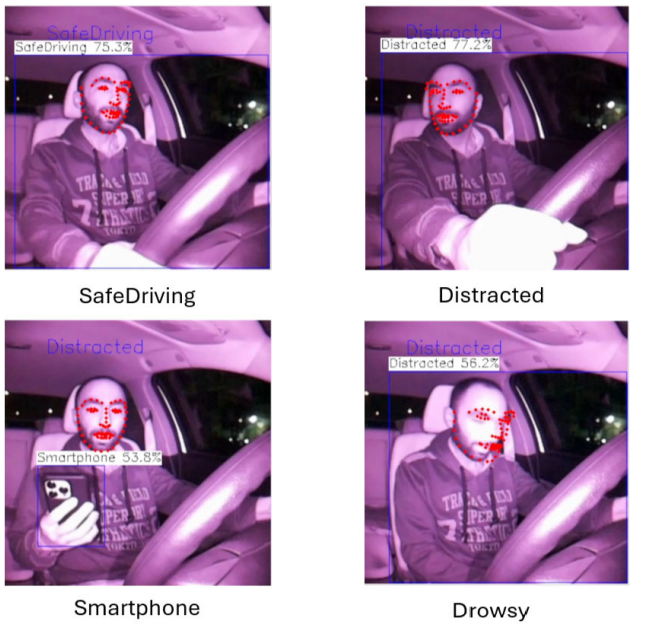

FIGURE 12. Trained model performance on testing data.

FIGURE 13. Real world testing setup with IR camera mounted in vehicle.

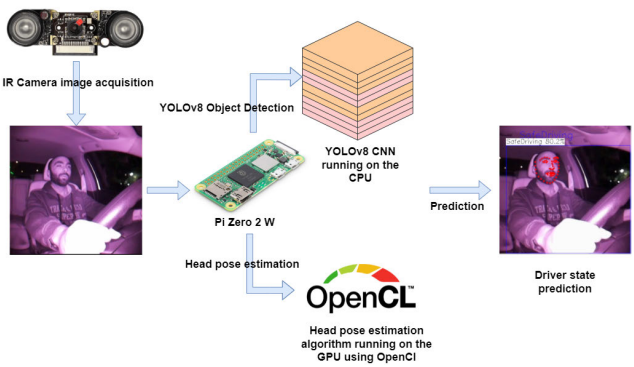

FIGURE 14. Overview of the proposed driver monitoring system.

Smartphone), instead of relying on the commonly used eye aspect ratio methods. The system uses YOLOv8 CNN for its high accuracy and performance on low-powered hardware. The $15 Raspberry Pi Zero 2 W is used as the main embedded hardware to deploy our model, compared with previous research methods that use a Raspberry Pi board shown in Table 9, this paper is the first to utilize the Raspberry Pi's GPU.

The Raspberry Pi's GPU is used to run a head pose estimation algorithm which is able to distinguish whether the driver is in a safe driving state or a distracted state. OpenCL is used to run the algorithm code on the Raspberry Pi GPU. It's worth noting that our algorithm does not fully exploit the GPU's potential, leaving room for processing additional data. For instance, sensor readings from the vehicle could be integrated into the driver monitoring system. YOLOv8 with GhostNet object detection model was trained on a dataset of more than 20000 images, resulting in mAP@50 over 90% with testing data and object detection accuracy on real-word testing data above 70%. The proposed system is developed with an IR camera to eliminate the negative effect of different lighting condition on model performance, the system is capable of running in daytime and nighttime, and was able to detect the driver's state in low-light conditions, an overview of the proposed system is shown in Fig. 14.

Combining YOLOv8 object detection running on the Raspberry Pi's CPU and head pose estimation algorithm running on the Raspberry Pi's GPU; the system is capable of monitoring the driver state while being cheaper than other methods that use the Raspberry Pi 3 or newer boards.

The developed system performed with an inference rate of 10 frames per second (FPS), According to the data presented by 100 Car Naturalistic Driving Study, eye glances away from the road for two seconds and longer doubled the risk of a crash or near crash [43], and the United Nation Economic Commission for Europe (UNECE) new regulation proposal for ADAS specify that driver monitoring systems shall warn the driver when distracted for longer than 5 seconds [44]. Our system can capture 20 frames within this critical 2-second window, and it can capture 50 frames in the 5-second window, allowing for timely distraction detection and response. It is important to note that this may not be sufficient for comprehensive crash detection in all scenarios. Future research is needed to evaluate the effectiveness of this frame rate in all possible real-world conditions and to explore potential frame rate enhancements.

V. CONCLUSION

In this paper, a low-cost driver monitoring system is presented, utilizing YOLOv8 CNN with a $15 Raspberry Pi Zero 2 W board for object detection of 4 different classes (SafeDriving, Distracted, Drowsy, and Smartphone) while also running a head pose estimation algorithm on the Raspberry Pi GPU using OpenCL to boost the accuracy of the driver state detection. Our trained model had an mAP50 of 90% and 10 fps inference rate. The proposed system is capable of monitoring the driver during both daytime and nighttime, as it relies on an IR camera to mitigate the negative effects of poor lighting conditions on the model's accuracy. The developed model struggled to detect smartphone usage in training and testing. Therefore, future research efforts will include developing a larger dataset from new distributions while focusing on smartphone detection performance, as well as taking advantage of the processing power still left available on the Raspberry Pi GPU by integrating vehicle sensor data analysis or adding another neural network. We also plan to deploy the model in vehicles and test methods for alerting the driver.

REFERENCES

[1] A. G. Wheaton, "Drowsy driving and risk behaviors—10 states and Puerto Rico 2011–2012," *MMWR Morb Mortal Wkly Rep*, vol. 63, no. 26, pp. 557–562, 2014.

[2] *NHTSA*. Accessed: Feb. 27, 2024. [Online]. Available: https://www.nhtsa.gov/risky-driving/drowsy-driving

[3] *Canalys*. Accessed: Feb. 20, 2024. [Online]. Available: https://www.canalys.com/newsroom/canalys-autonomous-driving-starts-to-hit-mainstream-as-35-million-new-cars-had-level-2-features-in-q4-2020

[4] O. Carsten, F. C. H. Lai, Y. Barnard, A. H. Jamson, and N. Merat, "Control task substitution in semiautomated driving: Does it matter what aspects are automated?" *Hum. Factors, J. Hum. Factors Ergonom. Soc.*, vol. 54, no. 5, pp. 747–761, Oct. 2012, doi: 10.1177/0018720812460246.

[5] J. W. Jenness, N. D. Lerner, S. Mazor, J. Osberg, and S. Tefft, "Use of advanced in-vehicle technology by young and older early adopters. Survey results on adaptive cruise control systems," NHTSA, U.S. Dept. Transp., Washington, DC, USA, Tech. Rep. DOT HS 810 917, 2008.

[6] *IDTechEx*. Accessed: Feb. 15, 2024. [Online]. Available: https://www.idtechex.com/ja/research-article/regulations-drivers-for-mandating-driver-monitoring-systems/30322

[7] *U.S. Congress*. Accessed: Feb. 15, 2024. [Online]. Available: https://www.congress.gov/bill/117th-congress/senate-bill/1406/text

[8] M. Q. Khan and S. Lee, "A comprehensive survey of driving monitoring and assistance systems," *Sensors*, vol. 19, no. 11, p. 2574, Jun. 2019, doi: 10.3390/s19112574.

[9] S. Titare, S. Chinchghare, and K. N. Hande, "Driver drowsiness detection and alert system," *Int. J. Sci. Res. Comput. Sci., Eng. Inf. Technol.*, vol. 7, pp. 583–588, Jun. 2021, doi: 10.32628/cseit2173171.

[10] C. Solomon and Z. Wang, "Driver attention and behavior detection with Kinect," *J. Image Graph.*, vol. 3, no. 2, pp. 1–6, 2015. [Online]. Available: https://www.joig.net/index.php?m=content&c=index&a=show&catid=42&id=106

[11] J. D. Ortega, N. Köse, P. N. Cañas, M.-A. Chao, A. Unnervik, M. Nieto, O. Otaegui, and L. Salgado, "DMD: A large-scale multi-modal driver monitoring dataset for attention and alertness analysis," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Jan. 2020, pp. 387–405, doi: 10.1007/978-3-030-66823-5_23.

[12] 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, doi: 10.1109/ACCESS.2020.2963960.

[13] E. Vural, "Drowsy driver detection through facial movement analysis," in *Proc. Int. Workshop Hum.-Comput. Interact.*, vol. 4796. Cham, Switzerland: Springer, pp. 6–18, doi: 10.1007/978-3-540-75773-3_2.

[14] K. Dwivedi, K. Biswaranjan, and A. Sethi, "Drowsy driver detection using representation learning," in *Proc. IEEE Int. Advance Comput. Conf. (IACC)*, Feb. 2014, pp. 995–999, doi: 10.1109/IADCC.2014.6779459.

[15] W. Zhang, Y. L. Murphey, T. Wang, and Q. Xu, "Driver yawning detection based on deep convolutional neural learning and robust nose tracking," in *Proc. IJCNN*, Jul. 2015, pp. 1–8, doi: 10.1109/ijcnn.2015.7280566.

[16] Y. Ji, S. Wang, Y. Zhao, J. Wei, and Y. Lu, "Fatigue state detection based on multi-index fusion and state recognition network," *IEEE Access*, vol. 7, pp. 64136–64147, 2019, doi: 10.1109/ACCESS.2019.2917382.

[17] K. Zhang, S. Wang, N. Jia, L. Zhao, C. Han, and L. Li, "Integrating visual large language model and reasoning chain for driver behavior analysis and risk assessment," *Accident Anal. Prevention*, vol. 198, Apr. 2024, Art. no. 107497, doi: 10.1016/j.aap.2024.107497.

[18] J. Chen, S. Dey, L. Wang, N. Bi, and P. Liu, "Attention-based multi-modal multi-view fusion approach for driver facial expression recognition," *IEEE Access*, vol. 12, pp. 137203–137221, 2024, doi: 10.1109/ACCESS.2024.3462352.

[19] J.-W. Kim, C. I. Nwakanma, D.-S. Kim, and J.-M. Lee, "Intelligent face recognition on the edge computing using neuromorphic technology," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2021, pp. 514–516, doi: 10.1109/ICOIN50884.2021.9333967.

[20] S. Y. Nikouei, Y. Chen, S. Song, R. Xu, B.-Y. Choi, and T. Faughnan, "Smart surveillance as an edge network service: From harr-cascade, SVM to a lightweight CNN," in *Proc. IEEE 4th Int. Conf. Collaboration Internet Comput. (CIC)*, Oct. 2018, pp. 256–265, doi: 10.1109/CIC.2018.00042.

[21] K. Biondi, E. Al-Masri, O. Baiocchi, S. Jeyaraman, E. Pospisil, G. Boyer, and C. P. de Souza, "Air pollution detection system using edge computing," in *Proc. Int. Conf. Eng. Appl. (ICEA)*, Jul. 2019, pp. 1–6, doi: 10.1109/CEAP.2019.8883458.

[22] J. Hariharan, R. R. Varior, and S. Karunakaran, "Real-time driver monitoring systems on edge AI device," 2023, *arXiv:2304.01555*.

[23] W. Kim, W.-S. Jung, and H. K. Choi, "Lightweight driver monitoring system based on multi-task mobilenets," *Sensors*, vol. 19, no. 14, p. 3200, Jul. 2019, doi: 10.3390/s19143200.

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[25] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713, doi: 10.1109/CVPR.2018.00286.
[26] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 6, pp. 1–50, Dec. 2023.
[27] N. N. Alajlan and D. M. Ibrahim, "DDD TinyML: A TinyML-based driver drowsiness detection model using deep learning," *Sensors*, vol. 23, no. 12, p. 5696, Jun. 2023, doi: 10.3390/s23125696.
[28] A. Chellappa, M. S. Reddy, R. Ezhilarasie, S. Kanimozhi Suguna, and A. Umamakeswari, "Fatigue detection using raspberry Pi 3," *Int. J. Eng. Technol.*, vol. 7, no. 2, p. 29, Apr. 2018, doi: 10.14419/ijet.v7i2.24.11993.
[29] J. Y. Wong and P. Y. Lau, "Real-time driver alert system using raspberry Pi," *ECTI Trans. Electr. Eng., Electron., Commun.*, vol. 17, no. 2, pp. 193–203, Aug. 2019, doi: 10.37936/ecti-eec.2019172.215488.
[30] M. Y. Hossain and F. P. George, "IoT based real-time drowsy driving detection system for the prevention of road accidents," in *Proc. Int. Conf. Intell. Informat. Biomed. Sci. (ICIIBMS)*, vol. 3, Oct. 2018, pp. 190–195, doi: 10.1109/ICIIBMS.2018.8550026.
[31] Daniel Stadelmann Thesis. *Entwicklung Einer OpenCL-Implementierung Fr Die VideoCore IV GPU Des Raspberry Pi*. Accessed: Dec. 15, 2023 [Online]. Available: https://github.com/doe300/VC4C/blob/master/doc/thesis.pdf
[32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.
[33] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Learn. Knowl. Extraction*, vol. 5, no. 4, pp. 1680–1716, Nov. 2023, doi: 10.3390/make5040083.
[34] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "GhostNet: More features from cheap operations," 2019, *arXiv:1911.11907*.
[35] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1867–1874, doi: 10.1109/CVPR.2014.241.
[36] S. Mohanty, S. V. Hegde, S. Prasad, and J. Manikandan, "Design of real-time drowsiness detection system using dlib," in *Proc. IEEE Int. WIE Conf. Electr. Comput. Eng. (WIECON-ECE)*, Nov. 2019, pp. 1–4, doi: 10.1109/WIECON-ECE48653.2019.9019910.
[37] W. H. Al-Sabban, "Real-time driver drowsiness detection system using dlib based on driver eye/mouth monitoring technology," *Commun. Math. Appl.*, vol. 13, no. 2, pp. 807–822, Aug. 2022, doi: 10.26713/cma.v13i2.2034.
[38] A. Pondit, A. Dey, and A. Das, "Real-time driver monitoring system based on visual cues," in *Proc. 6th Int. Conf. Interact. Digit. Media (ICIDM)*, Dec. 2020, pp. 1–6, doi: 10.1109/ICIDM51048.2020.9339604.
[39] X. X. Lu, "A review of solutions for perspective-n-point problem in camera pose estimation," *J. Phys., Conf. Ser.*, vol. 1087, Sep. 2018, Art. no. 052009, doi: 10.1088/1742-6596/1087/5/052009.
[40] *Roboflow Driver Monitoring Dataset*. Accessed: Sep. 1, 2024. [Online]. Available: https://universe.roboflow.com/dmd-test/dm-sys-dataset
[41] T. Carneiro, R. V. M. Da NóBrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance analysis of Google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, vol. 6, pp. 61677–61685, 2018, doi: 10.1109/ACCESS.2018.2874767.
[42] *YOLOv8 Project Github*. Accessed: Sep. 1, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics
[43] B. G. Simons-Morton, F. Guo, S. G. Klauer, J. P. Ehsani, and A. K. Pradhan, "Keep your eyes on the road: Young driver crash risk increases according to duration of distraction," *J. Adolescent Health*, vol. 54, no. 5, pp. S61–S67, Apr. 2014, doi: 10.1016/j.jadohealth.2013.11.021.
[44] *UN Regulation Proposal for ADAS*. Accessed: Sep. 1, 2024. [Online]. Available: https://unece.org/sites/default/files/2023-12/GRVA-18-07e.pdf

HADY A. KHALIL received the B.Sc. degree in mechatronics engineering from Ain Shams University, Cairo, Egypt, in 2020, where he is currently pursuing the M.Sc. degree. Since 2021, he has been a Software Development Engineer with Garraio for Software Innovations, Egypt, working on modern EVs development and embedded systems. His research interests include autonomous robotics, machine learning, embedded systems, automotive technology, and ADAS.

SHERIF A. HAMMAD was the Head of the Department of Mechatronics Engineering and the Dean of the School of Engineering, Ain Shams University. He is currently the Founder and the CEO at Garraio for Software Innovations, an embedded automotive software supplier to OEM and Tier 1. He is also a Professor of computer and systems. He served as the Minister of Scientific Research of Egypt.

HOSSAM E. ABD EL MUNIM received the B.Sc. and M.S. degrees in electrical engineering from Ain Shams University, Cairo, Egypt, in 1995 and 2000, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Louisville, Louisville, KY, USA. In June 2002, he joined the Computer Vision and Image Processing Laboratory (CVIP Lab), University of Louisville, where he has been involved in the applications of image processing and computer vision for medical image analysis. He is currently a Full Professor at the Department of Computer and Systems Engineering, Faculty of Engineering, Ain Shams University. His current research interests include image modeling, image segmentation, and deep learning, about which he has authored or co-authored more than 85 technical articles, including ICIP, MICCAI, ICCV, CVPR, BMVC, DICTA, IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, and *Journal of Real-Time Image Processing*.

SHADY A. MAGED received the B.Sc. and M.Sc. degrees in mechatronics from Ain Shams University, Cairo, Egypt, in 2010 and 2013, respectively, and the Ph.D. degree in mechatronics and robotics engineering with Egypt–Japan University for Science and Technology, in 2017. From 2010 to 2014, he was a Research and Lecturer Assistant with Ain Shams University, Egypt. In 2016, he was a Visiting Researcher with the Namerikawa Laboratory, Keio University. In 2017, he joined the Department of Mechatronics Engineering, Ain Shams University, where he has been an Associate Professor, since 2022. His research interests include the advanced model-based and intelligent control systems, system dynamics, robotics, soft robotics, and automation.