# Drivers Attention Detection Based on Quantized Convolutional Neural Network

Fernando Gutierrez-Herrera, Irene Gomez Jimenez, Alma Y. Alanis, and Gabriel Martinez-Soltero

Centro Universitario de Ciencias Exactas e Ingenierías, Universidad de Guadalajara, Guadalajara 44430, Mexico
erasmo.martinez@academicos.udg.mx

**Abstract.** There are a lot of reasons why a road accident happens, including road conditions, weather, car failures and human errors among others, that aren't easy to avoid or prevent, that's why in the last decade the technology has gained a main role when it comes to avoid these accidents as the Advance Drivers Assistance System or ADAS, where some of these technologies try to minimize road accidents due to lack of attention of the drivers on the road by detecting whether the driver is drowsy or distracted and notify him. However, not all vehicles are equipped with such systems. This paper presents a cost-effective and straightforward solution by implementing a quantized Convolutional Neural Network (CNN) on a low-cost module, achieving an accuracy of 86.7% of accuracy.

**Keywords:** CNN · Drivers Attention · Low-cost

## 1 Introduction

Whenever a traffic accident occurs, it not only puts the driver in danger, but it also puts those around them at risk, including other drivers, passengers, pedestrians, and cyclists. According to the 2017 Annual Road Safety Report, traffic accidents claim approximately 1.3 million lives worldwide each year, and an estimated 50 million people suffer serious injuries. Among the most vulnerable victims are the elderly, pedestrians, cyclists and motorcyclists [5].

Over the last decade, technology has taken a main role in preventing road accidents, a clear example of this is the Advance Driver Assistance System, or ADAS for short, which consists of the integration of sensors, mechanisms, algorithms, and even machine learning to give feedback to the driver about possible dangers or even take the control of the car to prevent these dangers [6][8].

In the field of machine learning, convolutional neural networks (CNN) are powerful algorithms for extracting features from structured data, particularly images. They are very effective in detecting and classifying objects, making them applicable to a wide range of fields, including automation, facial recognition, home security, and mores [7]. Since the introduction of classical CNNs, newer and more advanced architectures such as LeNet-5, AlexNet, VGGNet, and ResNet

https://doi.org/10.61728/AE20255336

have significantly improved the efficiency, performance, and accuracy in image recognition [1]. These advances have contributed to the increasing importance of object recognition in various real-world applications over the years.

The performance of CNNs depends not only on the training process but also on the hardware specifications they run on. CNNs can be deployed on various types of hardware, including high-performance GPUs, desktop or high-end CPUs, edge/mobile GPUs, FPGAs, and microcontroller units (MCUs), each with its own advantages and limitations [2]. While many hardware alternatives, such as high-end GPUs, desktop computers, and even less powerful microprocessors like the Raspberry Pi, are capable of handling large and complex CNN models, they can be unnecessarily expensive and impractical for certain applications. Running CNN models on microcontrollers presents a challenge due to their hardware limitations; however, it is still feasible. Despite this, microcontrollers are often regarded as a non-viable alternative for running high-performance AI models like CNNs due to their constrained processing power and different intended use cases.

Thanks to open-source machine learning frameworks like TensorFlow Lite Micro, it is now possible to run AI models on microcontrollers, even though their capacity and processing power are significantly lower compared to other types of hardware [3]. The ESP32 is one of the most suitable options for this purpose, primarily due to its hardware capabilities and the use of quantization techniques to optimize AI models for low-power execution. However, despite its potential, the integration of AI into microcontrollers has yet to be widely applied to critical real-world scenarios, such as driver drowsiness and attention detection. This is why deploying a CNN on a low-cost microcontroller equipped with a camera sensor, such as the ESP32-CAM, could provide a practical solution to alert drivers when they become distracted or fatigued. Additionally, it could demonstrate the feasibility of integrating AI with microcontrollers, expanding their application in safety and real-time monitoring systems.

This paper aims to review existing AI-based alternatives and projects designed to address real-world problems, particularly car accidents caused by human errors such as distraction or drowsiness while driving. It will compare the setup requirements and effectiveness of these solutions and explore how the ESP32-CAM, combined with a CNN model, could serve as a viable alternative. Additionally, the paper will detail the development process for locally running a pre-trained CNN model on an ESP32-CAM, covering the tools and techniques used. Finally, it will present the overall testing results, evaluate the performance of the system, and discuss potential improvements for future iterations of the project.

## 2 Related Work

One of the most versatile sensors used in ADAS is the camera, which can monitor both the vehicle's surroundings and its interior. In some cases, cameras are also used to detect driver behavior, such as eye blinking or head movements, to

determine whether the driver is distracted and needs to stop [6]. These and other technologies are primarily integrated by car manufacturers. While some external installations are relatively easy and affordable, others require significant modifications and come at a much higher cost.

When it comes to detecting driver drowsiness, there are three general approaches: vehicle-based, behavioral-based, and physiological-based. Among the proposed detection techniques are steering wheel movement monitoring using sensors, head tilt angle and blink rate analysis through cameras, and electroencephalogram (EEG) or heart rate monitoring [10]. Drowsiness detection should not only be non-intrusive but also ensure that the driver is effectively alerted without increasing the risk of an abrupt reaction—such as one caused by a sudden loud alarm—which could compromise safety.

There are ongoing studies and research efforts focused on developing new technologies and techniques to enhance Advanced Driver Assistance Systems (ADAS), particularly in estimating driver attention. These approaches analyze various factors, such as gaze direction and eye tracking, using eye shape detection and head pose estimation. To achieve this, multiple cameras and sensors are combined with machine learning algorithms, including fuzzy logic, neural networks, Bayesian networks, and supervised, semi-supervised, and unsupervised learning techniques, among others, to assess the driver's awareness of their surroundings [8]. While these technologies demonstrate significant potential, some face technical and practical limitations. For example, the use of sunglasses can interfere with eye tracking and shape detection, making it more challenging to accurately monitor driver attention.

There is a clear gap between a simple CNN running on a device like the ESP32 and the combination of advanced algorithms and AI techniques running on high-performance hardware. However, in terms of cost and accessibility, the ESP32-CAM presents a viable alternative for vehicles without built-in ADAS technology, providing an additional layer of safety while driving. With its easy setup and affordability, it offers a practical solution for enhancing driver awareness and road security.

## 3 Methodology

This section will outline the key steps and processes involved in the development of the project. It will include a brief review of the hardware and software components used, along with the tools selected. Additionally, it will cover the training process of the CNN model, the quantization of the model, and its deployment on an ESP32-CAM.

### 3.1 Hardware and Software Overview

The main component of this project is the ESP32-CAM module, which features a 2MP OmniVision OV2640 camera, as shown in Fig. 1. This module includes a built-in ESP32 microcontroller unit (MCU), which is powered by a

dual-core 32-bit processor running at 240 MHz. It has 448 KB of ROM, 520 KB of SRAM, and supports Wi-Fi (802.11n, 2.4 GHz, up to 150 Mbps) and Bluetooth (v4.2 BR/EDR and Bluetooth LE), among other features [4]. The ESP32-CAM was selected over other alternatives due to its hardware capabilities and ease of programming. For programming the module, the ESP32-CAM-MB board can be used, which includes a built-in CH340G USB-UART chip for serial communication via micro-USB/type-C. Alternatively, a general USB serial FTDI programmer, as shown in Fig. 2 can also be used. The data pre-processing, model architecture, training, validation, and quantization were conducted on a TensorFlow/Keras GPU-enabled workstation. The binary quantized model was generated by converting a TFLite file into a .h file using a Git-Bash command. Finally, the model deployment on the ESP32 was carried out using the MicroTFLite library within the Arduino [1] IDE.

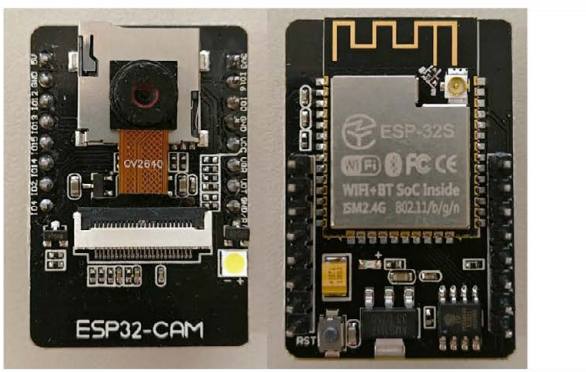


**Fig. 1.** ESP32-CAM Front and back.

**3.2 Data Pre-Processing And Model Architecture** A CNN model requires a dataset for training, so the first step was to collect representative data using the same hardware intended for testing. With the help of Arduino libraries, a dataset of 2,986 images was gathered. Leveraging the Wi-Fi capability of the ESP32-CAM, the microcontroller was connected to a home network, where the router assigned an IP address, allowing easy access from a PC. This setup enabled local image storage, with the images automatically categorized into three separate folders, each containing approximately 1,000 representative samples. The categories were labeled as focused driver, distracted driver, and drowsy driver, as illustrated in Fig. 3.

[1] Arduino is a registered trademark of Arduino AG. Hardware and software are available under open-source licenses. All rights reserved.

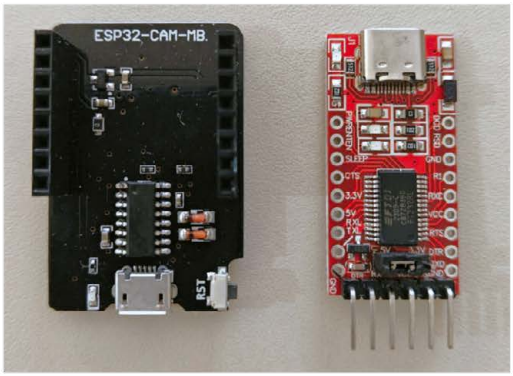


**Fig. 2.** Comunication modules.


**Fig. 3.** Set of labeled images taken from the train dataset.

With the collected representative data, a CNN model can be built and trained. To achieve this, a model architecture is required, either based on an existing design or a custom-made one developed using empirical criteria. The custom approach was chosen over pre-existing models to allow optimization for the limited memory space available on the MCU. The selected architecture is shown in Fig. 4. Before training, the dataset must be divided into three subsets to ensure proper training and validation: The largest portion of the dataset is allocated

for training, providing the model with sufficient examples to learn from. Half of the remaining dataset is used as a validation set during training. This subset is essential for evaluating the model's performance at the end of each epoch, helping to detect overfitting and guiding necessary hyperparameter adjustments.The other half is reserved for final validation, used to assess the overall performance of the model once training is complete.

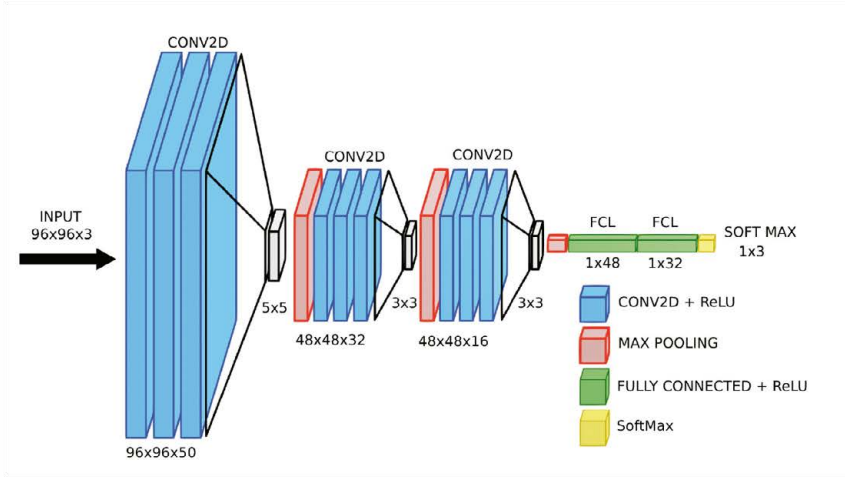


**Fig. 4.** Model Architecture (241,131 total parameters).

An additional technique to prevent overfitting is data augmentation, which involves generating new data from the existing dataset by applying reasonable modifications based on the study case. This process includes mirroring images along different axes, adjusting brightness, applying edge-enhancing filters, rotating images, and other transformations to increase the diversity of the training dataset. In Table 1 presents all the transformations applied to the training dataset, which initially comprised 2,090 images (70% of the original dataset). After applying data augmentation, the dataset size increased by 60%, resulting in a final training set of 3,344 images.

**Table 1.** Augmentation Transformations.

| Transformation Type | Range | Probability |
|---|---|---|
| Horizontal Flip | Y mirror | 60% |
| Random brightness and contrast | ±20% | 60% |
| Sharpen | 50% - 100% | 60% |

### 3.3 Model Training And quantization

The model was trained using 3,344 samples for training and 448 for validation, with a custom-scheduled learning rate for each epoch. Initially, a constant learning rate of 0.0001 was maintained for the first 38 epochs. In subsequent iterations, the learning rate was configured to decrease exponentially, scaled by a factor of 0.05, as an additional measure to prevent overfitting. To monitor model performance throughout training, a metric list was set up to track categorical cross-entropy loss and validation loss, alongside categorical accuracy, validation accuracy, recall, and precision. The loss function and accuracy trends over the epochs are presented in Fig. 5 and Fig. 6 respectively.

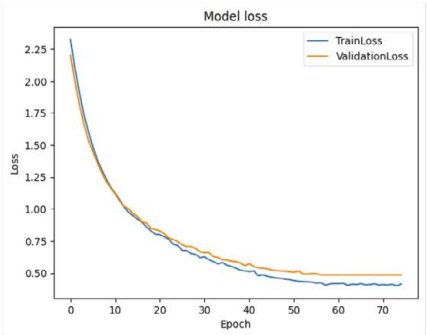


**Fig. 5.** Model loss and validation loss evolution per epoch.

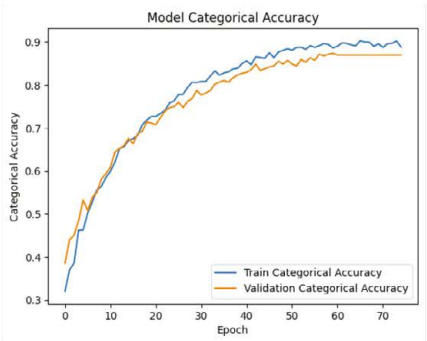


**Fig. 6.** Model categorical accuracy and validation categorical accuracy evolution per epoch.

The quantization process must be properly configured to generate a model compatible with the target device, in this case, the ESP32. Due to the ESP32's data management and processing limitations, it is necessary to define the input and output types of the quantized model, as well as the format in which operations are performed. In this case, all operations are set to int8 to optimize performance. Additionally, the custom quantization process for the ESP32 architecture requires a set of representative data from the test dataset to improve the model's efficiency. This step follows a procedure similar to validation in the original model training, ensuring the quantized model retains its accuracy while being optimized for deployment on the ESP32.

### 3.4 Quantized Model Deployment

Once the model is converted into a .tflite file, it must be transformed into a binary array to be loaded into the ESP32-CAM's memory. This can be accomplished using Git Bash with the following command: xxd -i model.tflite > model.h. This command generates a binary equivalent of the model, which can then be copied and pasted into a designated memory space on the ESP32. The model is then passed to the interpreter provided by the MicroTFLite library in the Arduino IDE for execution. The MicroTFLite library includes a basic example demonstrating its use with a simpler model that predicts the sine function directly on the ESP32. The implementation for this project was based on that example, adapting it for the CNN model.

## 4 Results

This section will evaluate the overall performance of both the original trained model and its quantized version. It will present the confusion matrix, along with an analysis to determine recall, precision, and F1-score for each category, as well as the overall accuracy of the quantized model based on validation dataset testing.

As a first approach to assess model performance, a small sample set from the test dataset was used to compare the model's predictions with their true labels. As shown in . 7 the model successfully predicted the correct labels for most of the sample set. Additionally, it demonstrated the ability to recognize diverse facial patterns, including different perspectives, hairstyles, the presence of glasses or hoods, and varying distances from the camera. This indicates good generalization capability. However, the model exhibited some confusion between the "drowsy" and "distracted" categories, highlighting the need for a confusion matrix analysis to further assess its classification accuracy and potential areas for improvement.

A confusion matrix is a graphical representation that illustrates the final performance of a model by comparing its predicted labels against the true labels from an unseen dataset that was not used during training. From this matrix, key performance metrics such as precision and recall can be derived. Precision represents the percentage of correct predictions relative to all instances recognized


**Fig. 7.** True and prediction comparing on a small set of samples from the test dataset.

by the model. Recall represents the percentage of correct predictions relative to all actual instances of that class in the dataset. Using precision and recall, a general performance metric, the F1-score, can be calculated. This metric provides a balanced evaluation of the model's accuracy, with a higher value (closer to 1) indicating better performance [9]. SSince the model classifies samples into one of three categories, these metrics must be computed for each class separately to assess how well the model distinguishes between them. The performance metrics for the model are presented in Table 2 based on the data from the confusion matrix in Fig. 8.

**Table 2.** Classification metrics.

| Category | Precision | Recall | F1-score |
|---|---|---|---|
| Attention | 0.888 | 0.931 | 0.909 |
| Drowzy | 0.882 | 0.789 | 0.833 |
| Distracted | 0.874 | 0.926 | 0.899 |

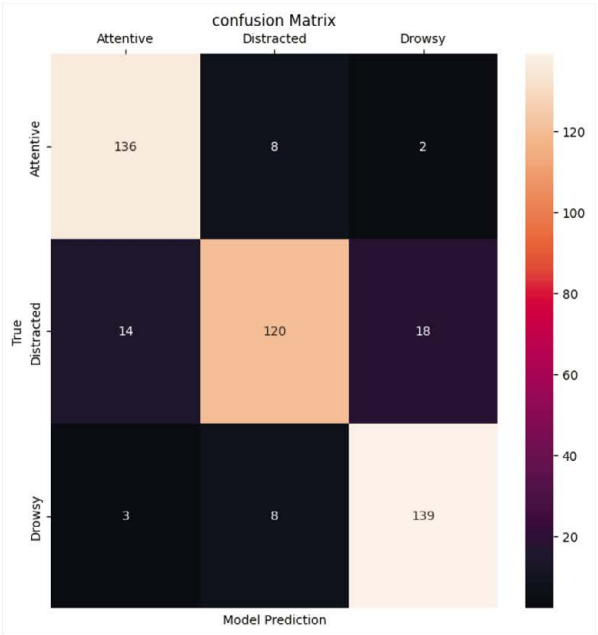


**Fig. 8.** Confusion matrix from test validation.

The performance evaluation of the quantized model was conducted by calculating its accuracy on the same test dataset used to assess the original model. The quantized model achieved an accuracy of 0.867. Based on this result, the model was deployed on the ESP32-CAM for real-world testing. Under similar conditions to those in the test dataset, the model's performance remained consistent, closely matching the calculated accuracy.

## 5 Conclusion

This study tested the implementation of quantized models on microcontrollers to evaluate their potential as a cost-effective and versatile alternative to certain ADAS technologies for inferring driver attention status. The proposed model achieved an 86.7% accuracy on unseen data, and the deployed version running locally on the ESP32-CAM demonstrated performance close to the estimated accuracy. For future improvements, considerations will include expanding the dataset to increase diversity and size, as well as exploring alternative model architectures to further enhance robustness and improve the reliability of the system.

## References

1. Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., Miao, Y.: Review of image classification algorithms based on convolutional neural networks. Remote Sensing **13**(22), 4712 (2021)
2. Chitty-Venkata, K.T., Somani, A.K.: Neural architecture search survey: A hardware perspective. ACM Computing Surveys **55**(4), 1–36 (2022)
3. David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T., et al.: Tensorflow lite micro: Embedded machine learning for tinyml systems. Proceedings of Machine Learning and Systems **3**, 800–811 (2021)
4. Dietz, H., Abney, D., Eberhart, P., Santini, N., Davis, W., Wilson, E., McKenzie, M.: Esp32-cam as a programmable camera research platform. Imaging **232**(2), 10–2352 (2022)
5. Janstrup, K.H.: Road safety annual report 2017 (2017)
6. Jumaa, B.A., Abdulhassan, A.M., Abdulhassan, A.M.: Advanced driver assistance system (adas): A review of systems and technologies. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) **8**(6), 231–4 (2019)
7. Kadhim, T.A., Hariri, W., Smaoui Zghal, N., Ben Aissa, D.: A face recognition application for alzheimer's patients using esp32-cam and raspberry pi. Journal of Real-Time Image Processing **20**(5), 100 (2023)
8. Khan, M.Q., Lee, S.: Gaze and eye tracking: Techniques and applications in adas. Sensors **19**(24), 5540 (2019)
9. Nemavhola, A., Chibaya, C., Viriri, S.: A systematic review of cnn architectures, databases, performance metrics, and applications in face recognition. Information **16**(2), 107 (2025)
10. Saini, V., Saini, R.: Driver drowsiness detection system and techniques: a review. International Journal of Computer Science and Information Technologies **5**(3), 4245–4249 (2014)