MDPI

Article

Driver Monitoring System with Arduino Nano 33 BLE

Harsh Gangwar, Rohan khaire, Joel Eliston and Rohit Menon

[1] hgangwar@asu.edu
[2] rkhaire@asu.edu
[3] jeliston@asu.edu
[4] rmenon12@asu.edu

Abstract: This paper presents the development of a Driver Monitoring System (DMS) utilizing the Arduino Nano 33 BLE and an integrated camera module. The system focuses on real-time monitoring of the driver's behavior using Convolutional Neural Networks (CNNs). The Arduino Nano 33 BLE serves as a compact and low-power microcontroller and its camera captures real-time images of the driver. The CNN model is employed to analyze facial expressions, eye movements, and head gestures, providing a comprehensive assessment of the driver's attention and alertness levels. The system aims to enhance road safety by detecting signs of drowsiness or distraction, and subsequently alerting the driver in real-time. The utilization of Arduino Nano 33 BLE ensures a cost-effective and easily deployable solution, making it suitable for integration into various vehicles. The real-time CNN-based analysis enables quick and accurate identification of potential driver safety risks, contributing to a proactive approach in preventing accidents caused by driver-related factors. The proposed Driver Monitoring System offers a versatile and accessible solution for enhancing driver safety through continuous monitoring, paving the way for the integration of advanced driver-assistance systems in a wide range of vehicles..

Keywords: Deep learning; Driver fatigue detect; Embedded system; Image processing

1. Introduction

Our research addresses a significant gap in the current landscape where high-end Python or Java-based products dominate the industry. The primary goal is to democratize access to advanced technology by creating a cost-effective and affordable solution accessible to a broader audience. Focusing on low-cost microcontrollers, our work explores the boundaries of complexity in model design, the scalability of label consideration, and the identification of the most suitable CNN architecture for this application.

In this pursuit, we aim to redefine the parameters of feasibility and effectiveness in the realm of cost-effective solutions. By pushing the boundaries of what low-cost microcontrollers can achieve, we seek to determine the optimal balance between affordability and performance. Our research delves into the intricacies of deploying complex models in resource-constrained environments and addresses critical questions about the number of labels that can be accommodated. Through this investigation, we aspire to provide comprehensive answers that contribute to the development of economically viable solutions, making advanced technology accessible to a wider demographic and fostering innovation in the integration of affordable, high-performance systems.

2. Related Work

The current state of the art in monitoring systems includes noteworthy developments by industry leaders such as NXP Semiconductor and Qualcomm, both showcasing advanced Driver Monitoring Systems (DMS). NXP's system is designed around five features, including Yawning, Drinking, Smoking, Eyes closed, and On a call. Meanwhile, Qualcomm's DMS, presented at CVPR23, exhibits a broader capability, detecting activities such

Citation: Gangwar, H.; Khaire, R.; Eliston, J; Menon R. Title. *Journal Not Specified* **2023**, *1*, 0. https://doi.org/

Received:
Revised:
Accepted:
Published:

as Calling, Drinking, Smoking, Yawning, Hands off steering wheel, Arm out windows, and Looking direction.

However, a literature gap emerges concerning cost-effective solutions and the integration of such advanced monitoring capabilities into low-cost microcontrollers. Notably, existing research has explored Embedded systems using Nvidia Jetson Nano and Jetson Tk1 for DMS, with a focus on detecting Eye closing and yawning.

In this context, our work aims to bridge this gap by bringing the capabilities of sophisticated monitoring systems to the accessible and cost-effective realm of Arduino Nano 33 BLE. This shift addresses the need for affordable solutions without compromising the intricacies of detection, expanding the possibilities for implementing robust Driver Monitoring Systems in a wider range of applications and environments.

3. System Design

In crafting our system design, our approach is centered on a thorough exploration of different system architectures to gauge their effectiveness. Currently, we are in the initial phase of establishing benchmark algorithms/models to assess the performance of diverse architectures during experimentation.

For model training, we have devised a custom Convolutional Neural Network (CNN) architecture capable of recognizing 10 distinct classes as specified in the dataset. In addition to our custom CNN, we implemented transfer learning using pre-trained models like MobileNetV3 and DenseNet. Leveraging the transfer learning technique, we fine-tuned these models for our specific application.

Training was conducted on a local system equipped with an RTX 3070 GPU boasting 8GB of VRAM, ensuring efficient processing. The training results showcase the efficacy of our models, with the custom CNN achieving a training accuracy of 97% and a validation accuracy of 96.5%. MobileNetV3 exhibited a training accuracy of 98.3% and a validation accuracy of 96.7%, while DenseNet achieved a training accuracy of 93.3% and a validation accuracy of 92.1%.

As our work progresses, this foundation will guide us in selecting the most suitable system architecture for real-time execution on embedded devices, with a focus on optimizing the model for efficient and effective performance.

4. Evaluation Approach

In our exploration of driver monitoring system architectures, we present model accuracy and loss plots for three distinct models: Custom CNN, MobileNetV3, and DenseNet. These visualizations offer insights into the training process, showcasing the performance of each architecture.

4.1. Figures, Tables and Schemes

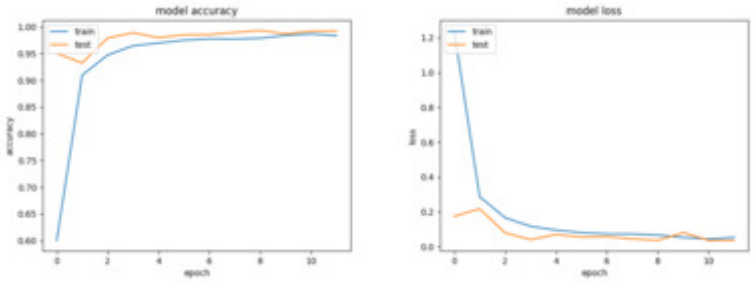

Figure 1. Custom CNN: (**a**) Model Accuracy. (**b**) Model Loss.

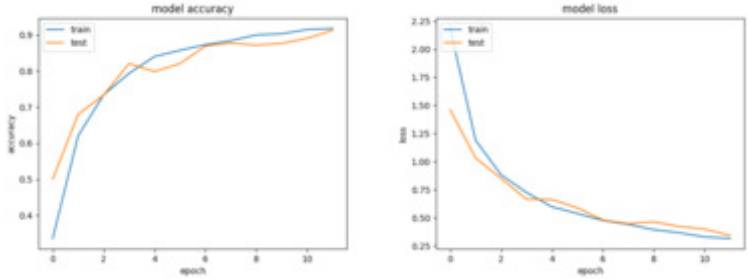

Figure 2. DenseNet: (**a**) Model Accuracy. (**b**) Model Loss.

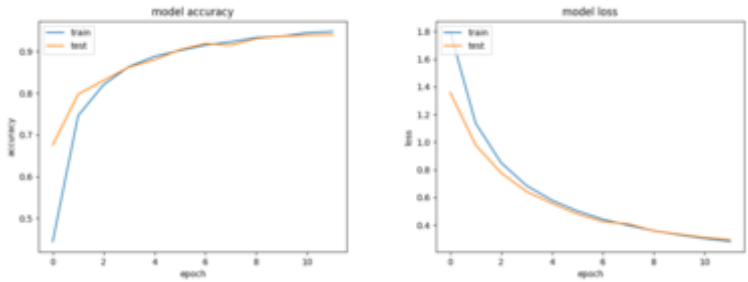

Figure 3. MobileNET V3: (**a**) Model Accuracy. (**b**) Model Loss.

5. Experimentation Process

5.1. Arduino

Initially, we adapted various models (including a custom Convolutional Neural Network (CNN), MobileNetV3, and DenseNet) to ensure compatibility with the Arduino platform. Subsequently, we modified the example code, specifically the one labeled "person detection" in Harvard Tiny ML, to repurpose it as a driver detection system.

During the course of the experiment, we encountered a challenge when our model turned out to be too large for the Arduino's capacity. To address this issue, we redesigned the model, adjusting the image size to 96x96 pixels and converting it to grayscale. This alteration allowed us to fit the model within the dynamic memory constraints of the Arduino. However, a new problem arose as the Arduino consistently generated the output "Invoke failed." We suspect that the model architecture we created differs from the one employed by Harvard Tiny ML. Despite extensive online research, we were unable to locate information on the specific model they utilized.

Faced with this roadblock, we made the decision to switch to a BeagleBone Black microcontroller. The BeagleBone's capability to operate its own Linux operating system provided us with confidence that our project could be successfully executed on this platform.

5.2. BeagleBone

To employ the BeagleBone, our initial step involved installing all the necessary libraries, with a particular emphasis on OpenCV. Initially, we attempted to build OpenCV directly on the BeagleBone. However, due to its constrained computing resources, we encountered an out-of-memory error during the build process. Subsequently, we opted for a cross-compilation approach, using a Docker image of Debian that emulates the BeagleBone's architecture (ARMHF). After successful compilation, we transferred the OpenCV library to the BeagleBone, effectively integrating it into the platform.

In our attempts to capture live images from a camera, we encountered a challenge when attempting to connect a webcam directly to the BeagleBone, as the microcontroller lacked the power to support it. To address this limitation, we explored an alternative approach by attempting to access the laptop's webcam through the BeagleBone. Unfortunately, we encountered permission issues that prevented successful access. Ultimately, we

devised a solution using a Python script to transmit the webcam feed through a socket, paired with another Python script on the BeagleBone to receive and display the live images. This workaround allowed us to successfully obtain and display live images on the BeagleBone.

Now, our project involves the use of two Python scripts to facilitate its operation.

The first script is responsible for capturing live video as bytes from the webcam connected to the laptop. Subsequently, it transmits this data to the BeagleBone, establishing a seamless communication channel between the devices.

The second script is designed to receive the transmitted data on the BeagleBone. It performs the essential task of converting the received bytes back into images. These images are then fed into our model, enabling it to generate predictions based on the input data.

This dual-script setup ensures the effective execution of our project, allowing for the seamless transfer and processing of live video data from the laptop to the BeagleBone, ultimately resulting in accurate predictions from our model

5.3. Neural Network Structure

Our neural network was trained using the MobileNetV1 architecture, utilizing pre-trained weights from ImageNet. To optimize memory efficiency for embedded devices, we applied a width factor of 0.25 during training. As a result, the model size was constrained to 875 kilobytes (kbs). Subsequently, to further enhance its suitability for embedded systems, we performed full 8-bit integer quantization on our original 32-bit float model. The quantized model, now represented in 8-bit integer format, was successfully compressed to a size of 300 kilobytes (kbs). This reduction in size ensures the efficient deployment of the neural network on resource-constrained devices while maintaining its predictive capabilities.

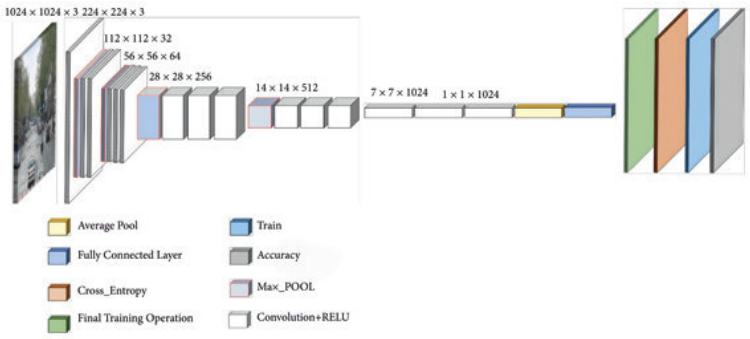

Figure 4. MobileNet V1 Architecture

5.4. Results

Ultimately, we opted to adhere to the MobileNet-derived model, as the other models proved to be incompatible. The achieved results demonstrate a validation accuracy of approximately 85% and a validation loss of around 0.18.

5.5. Figures, Tables and Schemes

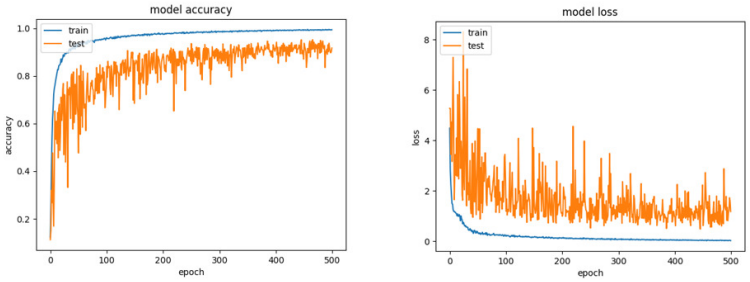

Figure 5. MobileNetV1: (**a**) Model Accuracy. (**b**) Model Loss.

For the specific requirements of our project, which involves binary classification into two labels (Alert and Not Alert), we made the following interpretation of the model's predictions: If the model predicted class "c0," it was interpreted as Alert, whereas predictions falling into any other classes (c1 – c9) were categorized as Not Alert. This simplified classification approach effectively met the needs of our project while maintaining a satisfactory level of accuracy.

References

1. 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," 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Honolulu, HI, USA, 2017, pp. 438-445, doi: 10.1109/CVPRW.2017.59. (Accessed: 08 November 2023).
2. Esra Civik, Ugur Yuzgec, Real-time driver fatigue detection system with deep learning on a low-cost embedded system, Microprocessors and Microsystems, Volume 99, 2023, 104851, ISSN 0141-9331, https://doi.org/10.1016/j.micpro.2023.104851 (https://www.sciencedirect.com/science/article/pii/S0141933123000972)(Accessed: 08 November 2023).