Driver Distraction Detection on Edge Devices via Explainable Artificial Intelligence

Mario G.C.A. CIMINO
Department of Information Engineering, University of Pisa
Pisa, 56122, Italy

Antonio DI TECCO
Department of Information Engineering, University of Florence, University of Pisa
Florence, 50139, Italy

Pier Francesco FOGLIA
Department of Information Engineering, University of Pisa
Pisa, 56122, Italy

Tommaso NOCCHI
Department of Information Engineering, University of Pisa
Pisa, 56122, Italy

Cosimo Antonio PRETE
Department of Information Engineering, University of Pisa
Pisa, 56122, Italy

Marco RALLI
Department of Information Engineering, University of Pisa
Pisa, 56122, Italy

Lorenzo TONELLI
Department of Information Engineering, University of Pisa
Pisa, 56122, Italy

ABSTRACT

Driver Attention Monitoring is a challenging research task, with several complex behavioral distractions to recognize, and the need to use non-invasive on-board systems. Recent advances in deep learning have great potential in this field. This research aims to propose an architectural solution based on a deep convolutional neural network, deployed on an edge device. For this purpose, a publicly available dataset has been exploited to recognize various distracting driver behaviors. The model has been validated using explainable AI techniques. Experimental studies show that the proposed architectural solution, deployed on an NVIDIA Jetson Nano board, achieves a throughput of 11 frames per second and an accuracy of 92%. In contrast to the recent state-of-the-art solutions, the proposed approach covers all the relevant requirements: (i) it is non-invasive; (ii) it has a complexity suitable for popular edge devices; (iii) it allows a reliable validation of the different driver distractions via explainable AI; (iv) it achieves a competitive accuracy.

Keywords: driver distraction detection, image recognition, edge device, convolutional neural network, explainable artificial intelligence.

1. INTRODUCTION

The proposed Driver Attention Monitoring System (DAMS) aims to efficiently classify driver behavior to detect distractions. In the last years, due to their non-invasiveness, computer vision techniques based on deep learning have become more promising than physiological-based and hybrid DAMS. Indeed, the former requires only a camera, whereas the latter requires that sensors are attached to the driver. On the other hand, management and execution efforts are important requirements for popular and resource-constrained systems such as vehicles.

The industry is highly interested in operational DAMS. For instance, manufacturers such as Volvo have recently developed safety systems in their car series [1]. Some of these systems use electric signals from mechanical aspects such as pressure on the pedals, angular acceleration during curves as well as hand placement. Although camera-based techniques are still in the experimental phase, Volvo recently introduced into production a hybrid system, i.e., combining mechanical signals with those detected by camera monitoring. These systems mainly work on image sequences that detect the driver's head position to determine the driver's status, trying to understand when the user is not focused on driving [6]. In recent years, the maturity achieved by Artificial Intelligence (AI) techniques has piqued interest into the automotive field. One promising feature of AI systems is the possibility to develop architectures for behavioral recognition, and in particular to recognize dangerous behaviors that distract the driver.

In the literature, this problem has been mainly considered from a proof-of-concept point of view: to demonstrate that a suitable deep neural network can achieve adequate accuracy. However, research works focused on how to deploy DAMS in constrained situations are still in the preliminary phase. The automotive sector, in fact, is characterized by energy, computational, and physical constraints that make it difficult to create and deploy complex models. A viable solution could be the integration of the model into a cloud system. However, this introduces a number of related problems in terms of security, privacy, and response time.

In this research work, a non-invasive DAMS based on camera and deep learning is developed and implemented on an edge system. The edge system is able to perform both image acquisition and behavioral recognition. Different architectural settings have been analyzed, to select the best model via different metrics. In order to validate the solution, Explainable AI techniques have been used to understand the behavior of the model during classification. The achieved accuracy is promising with respect to the state-of-the-art. In addition, the proposed DAMS exhibits a fast response time, while it is deployed in resource-constrained environment. Specifically, the system has been implemented on an edge device, namely a Raspberry 4B with hardware accelerators, and evaluated also on an NVIDIA Jetson Nano board, achieving the best performance. It can be integrated with industrial systems to give a complete view of what is happening inside the cockpit.

The paper is structured as follows. Section II focuses on related work. The dataset and its preprocessing are covered in Section III. In Section IV, the proposed architecture and its development are detailed. Experimental results are discussed in section V. Section VI summarizes the implementation aspects, whereas the driver's behavior detection pseudocode is studied in Section VII. A performance analysis of the implemented system is carried out in Section VIII. Finally, Section IX draws some conclusions.

2. RELATED WORK

This section is devoted to review the relevant work from the literature on driver state monitoring, with a focus on the use of convolutional neural networks fed by images taken by an in-dash camera.

A well-known example of distracting behavior is the use of a smartphone [2], [19], [20]. One of the first research works and datasets developed with images made through onboard cameras was published by Zhang *et al.* [3]. Specifically, the authors proposed an approach to detect the use of a mobile phone based on the hands, face and mouth features of the drivers. Another relevant research was developed in 2015, by Nikhil *et al.* [4]. The authors created a dataset for hand detection in the automotive environment, achieving an average precision of 70% via the Aggregate Channel Features (ACF) object detector. On the other hand, Le *et al.* [5] proposed a hand and face detector based on the Faster RCNN architecture, outperforming the existing methods by achieving an accuracy of 94.2%. Specifically, the proposed system was able to perform hands-on-wheel detection at a rate of 0.09 frames per seconds (*fps*), i.e., it required about 11 s to elaborate a frame. Another approach named Automatic Identification of Driver's Smartphone [18] was developed to find the position of the smartphone in the car, by analyzing and fusing information from sensors available on commodity smartphones.

In general, neural networks allow to develop systems capable of recognizing the driver's behavior with greater precision. Today, some works analyze this approach [6]. Different deep learning architectures are today available for this purpose: YOLOv4 [7], SSD [8], Faster-RCNN [9], and so on. An essential requirement concerns the dataset used for the creation of models, which must be based on images captured directly inside the car, under different environmental circumstances. However, several studies in the literature have investigated how various deep neural networks can classify images without considering their deployment in an embedded system for a car. With regard to this requirement, the work presented in [10] reports the analysis of some neural network architectures, such as VGG, Resnet, Alex Net, and Google Net, applied on a Jetson TX1 board, an embedded system-on-module (SoM) with quad-core ARM Cortex-A57, 4GB LPDDR4 and integrated 256-core Maxwell GPU. It is suitable for deploying computer vision and deep learning. The authors achieved an accuracy of around 86% via the faster neural network. In addition to the networks described above, other models can be considered. In [11], the authors analyzed the problem using a neural network architecture called VGG19. However, the system was developed on high-performance hardware: Intel i5 quad-core 3.5 GHz, RAM 16 GB, and NVIDIA GPU GTX 1070 8 GB. In contrast, the focus of this research is to develop a DAMS on an effective edge device.

Other approaches in the literature [12] are based on the analysis of the user's face, focusing on the perceived fatigue rather than on the type of distraction. The approach is based on image inspection, focusing on the mouth and eyes. The proposed solution does not require the use of artificial intelligence, but it is knowledge-based: it is mainly based on the use of analytical models processing portions of the face.

In a research work [13] Sahoo *et al.* developed a Squeeze Net neural network that classifies driver's images provided by a camera. The neural network can be loaded and executed directly on a Raspberry. A limitation of the proposed approach is that it classifies the driver through a single image (frame). In contrast, the approach proposed in this paper generates a classification based on a sequence of images, to better represent the driver's behavior. Another difference is on the dataset: in this paper, the dataset is enriched with images taken in the wild by the Raspberry Camera. The purpose is to improve the generalization capabilities of the network by considering additional real-life scenarios. The next Section focuses on the dataset preparation.

3. DATASET AND ITS PREPROCESSING

In order to cover the most relevant distracting driver's behavior, an initial dataset has been enriched. The initial dataset consists of a collection of images divided into 10 different classes, exemplified in Figure 1:

- C0: safe driving;
- C1: driver is texting using the right hand;
- C2: driver is talking on the phone using the right hand;
- C3: driver is texting using the left hand;
- C4: driver is talking on the phone using the left hand;
- C5: driver is operating the radio;
- C6: driver is drinking;
- C7: driver is reaching behind;
- C8: driver is tidying up hair or applying makeup;
- C9: driver is talking to passenger.

To differentiate the images of the dataset, the images are first subsampled, for selected drivers and for chromatic effects. Numerous cases of accidents are caused by distraction induced by activities related to mobile phone use [19], [20]. To give more focus on the use of the phone, the dataset is then divided into three different classes: safe driving, using the phone, distraction (e.g. operating the radio, or taking your eyes off the road momentarily).

Specifically, all images of classes C1, C2, C3, and C4 are categorized into a new class *Phone*, whereas images of classes C5, C6, C7, and C8 are categorized into a new class *Distraction*. Finally, Class C0 is categorized into a class *Safe*. It is worth noting that Class C9 has been ignored as it is similar to Class C0. Furthermore, to better generalize the network classification capability, the dataset has been enriched with novel in-car images captured in the wild by the Raspberry Pi Camera v2 (Figure 2).

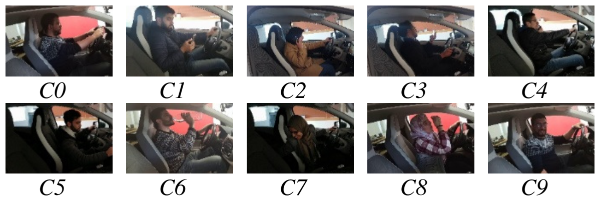


Figure 1: Images of Driver Distraction dataset [26].

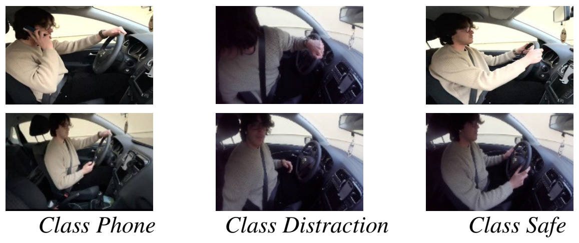


Figure 2: Images taken in the wild by Pi Camera.

In order to mitigate textural bias, data augmentation has been also applied to all classes, by carrying out a number of transformations such as flipping (horizontally and vertically) and rotation (90°, 180°, and 270°). To standardize the input size, all images have been resampled to 224×224 pixels. To build the final dataset, 3,261 images have been randomly selected: 1,172 for Class *Phone*, 986 for Class *Distraction*, and 1,103 frames for Class *Safe*. At this point, the dataset has been split into training, validation, and test sets. Each set with a different number of frames per class as summarized in Table 1.

Table 1: Dataset distribution per set and class.

Set	Phone	Dist.	Safe	Total
Training	987	826	881	2,694
Validation	104	85	130	319
Test	81	75	92	248
Total	1,172	986	1,103	3,261

4. PROPOSED ARCHITECTURE AND ITS IMPLEMENTATION

The architecture adopted in this work is the VGG16 deep neural network model [14], pre-trained with the ImageNet dataset [14]. We tuned the VGG16 via transfer-learning techniques [24]. The VGG16 is a convolutional model used for object detection and classification. Figure 3 shows the overall architecture of the VGG16 network. Specifically, the convolutional base consists of five blocks with either two or three convolutional layers, with 3×3 filters. All hidden layers are equipped with ReLU layers. Furthermore, max-pooling is performed on a 2×2 window between blocks, with stride 2. The convolutional base is followed by a classifier made of three fully connected layers. The final layer is a softmax layer. The network expects a fixed dimension input of 3×224×224, and 1,000 probability output values.

With respect to the standard VGG16 classifier, in the developed architecture the classification layer is comprised of two fully connected layers of 25,088 and 128 neurons, and a softmax layer, which outputs three probability values, corresponding to the three classes.

The training process has been based on two strategies:

(i) *feature extraction*: the VGG16 convolutional base is frozen, and feature extraction is carried out by taking its output for training the classification layer;
(ii) *fine-tuning*: the VGG16 convolutional base has been fine-tuned, by optimizing the convolutional blocks and freezing the others, to train the classification layer.

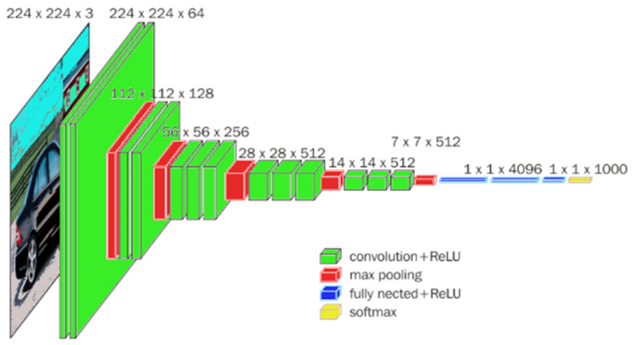


Figure 3: VGG16 model architecture [14].

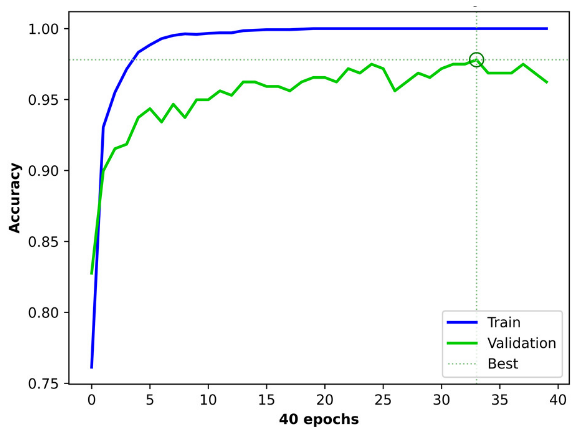
According to the two training strategies, three models have been trained: (i) *standard model*, trained with feature extraction; (ii) *first fine-tuned model*, trained via the fine-tuning strategy with 5th convolutional block optimized; (ii) *second fine-tuned model*, trained via the fine-tuning strategy with 5th and 4th convolutional block optimized.

The architectural implementation has been based on Python 3.9 64-bit [22] and the Pytorch package [23]. As a loss function, the *cross-entropy* has been used. As an adaptive parametric optimizer, the *RMSprop* algorithm has been used, with learning rate lr=10^{-5} and epsilon eps=10^{-7}. Early stop on validation set loss has been used to appropriately stop the training process, with patience equal to 6 consecutive epochs. The training has been carried out on an Intel CPU i7-12700K, 32 GB of DDR4 DRAM, and NVIDIA GPU 3060 Ti.

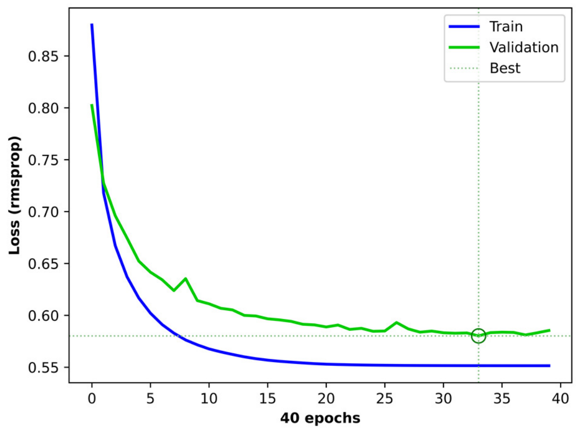
5. EXPERIMENTAL RESULTS

The different architectural models have been sequentially trained. In the following, some training processes are summarized and represented. The standard model achieved the best performance at epoch 33 with a validation loss of 0.5802 and an accuracy of 0.9781 (Figure 4). The first fine-tuned model achieved the best performance at epoch 15, with a validation loss of 0.5789 and an accuracy of 0.9749 (Figure 5). Finally, the second fine-tuned model achieved at epoch 26 a validation loss of 0.5728 with an accuracy of 0.9749 (Figure 6). As a matter of fact, the three modes achieve similar validation accuracy. In order to understand the generalization capabilities, the three models have been evaluated on the test set. Results are illustrated in terms of confusion matrixes in Figure 7 (standard model), Figure 8 (first fine-tuned model), and Figure 9 (second fine-tuned model).

In each confusion matrix, the last column reports, for each class: (i) the total number of outputs generated by the classifier; (ii) Precision rate [28], i.e., TP/(FP+TP), where TP and FP are the number of True Positive and False Positive samples, respectively. The last row reports, for each class: (i) the total number of inputs belonging to that class; (ii) Recall rate [28], i.e., TP/(TP+FN), where FN is the number of False Negative samples. Finally, the last box along the main diagonal represents the overall Accuracy, i.e., the fraction of correct classifications. A good classifier must have both precision and recall with high values [28], [29]: the precision represents the classifier's effectiveness in recognizing the class (proportion of positively classified identifiers that are correctly predicted), while the recall represents the classifier's ability to correctly detect samples belonging to the class (proportion of real positives that are correctly predicted).



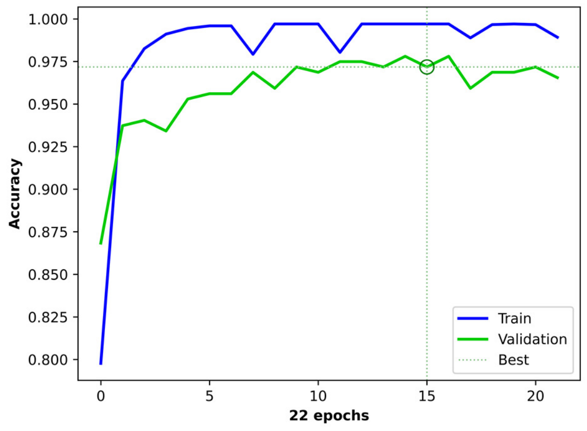
(a)



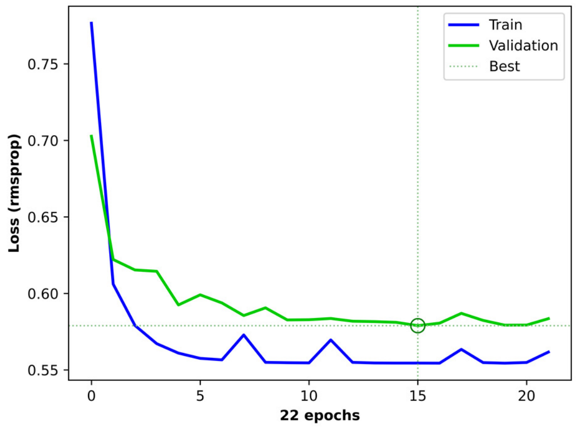
(b)

Figure 4: Standard model accuracy (a) and loss (b) for training and validation sets.

The number of correct classifications (True Positive) for each class is represented on confusion matrices by the values in the green boxes. The lowest value is achieved by the standard model (Figure 7). The first fine-tuned model achieves better results, in particular for classes Distraction and Safe (Figure 8). However, for this model, while the Recall rate increases or remains constant for each class, the Precision rate decreases for the phone class. In contrast, in the second fine-tuned model the number of correct classifications increases for each class, and the Precision and Recall rates also increase for each class (Figure 9). The better performance of the second fine-tuned model is also confirmed in Table 2, where the overall model accuracy and weighted F1-score, i.e., the harmonic mean of Precision and Recall rates, are shown. Specifically, the model with the best accuracy is the second fine-tuned model, which achieves an accuracy of 0.9234.



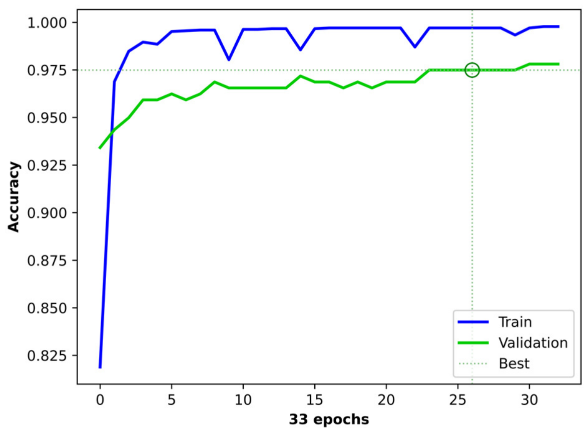
(a)



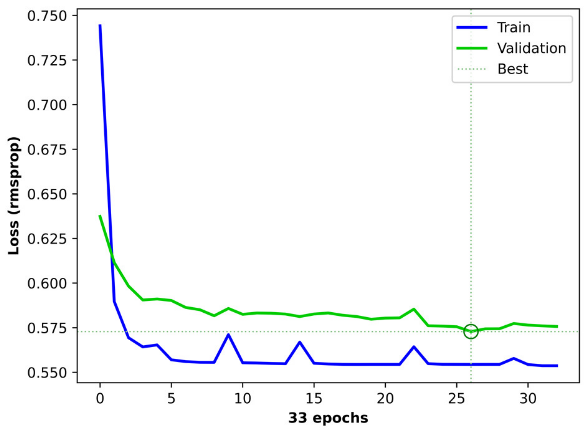
(b)

Figure 5: First fine-tuned model accuracy (a) and loss (b) for training and validation sets.

In order to understand how the best-performing model classifies images, the class activation heatmaps have been generated [27]. This technique is useful to understand which parts of a given image (represented in hot colors) led a convolutional model to its classification output [27]. To this aim, Figures 10-12 show some pairs of sample images to compare. For each pair, the first image is correctly classified, whereas the second one is erroneously classified. Specifically, Figure 10 represents safe driving: both heatmaps focus on the driver's hand positions, correctly kept on the wheel, and directed towards the road; however, the second heatmap is partially activated by the front panel, which has been confused with a phone. On the other hand, Figure 11 represents distracted driving: the driver is taking his hands off the wheel, rotating his body in Figure 11a; however, in Figure 11b the heatmap is partially activated by the external environment, leading to the class Phone. Finally, Figure 12 shows a driver who is talking on or picking up the phone: the heatmap in Figure 12a correctly focuses on the driver's right hand, which is holding the phone; however, the heatmap in Figure 12b does not recognize the phone in the driver's left hand, but it focuses on the free steering, giving the Distraction class.

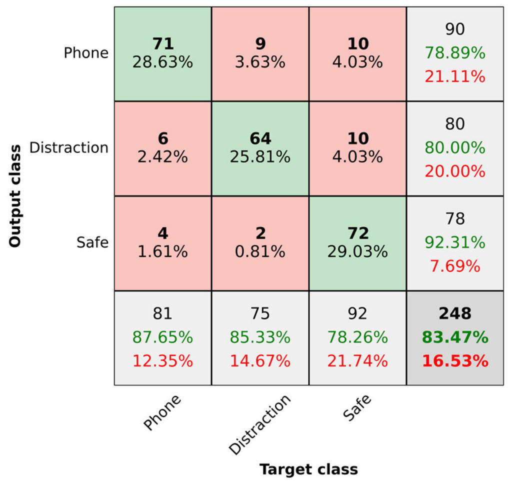


(a)



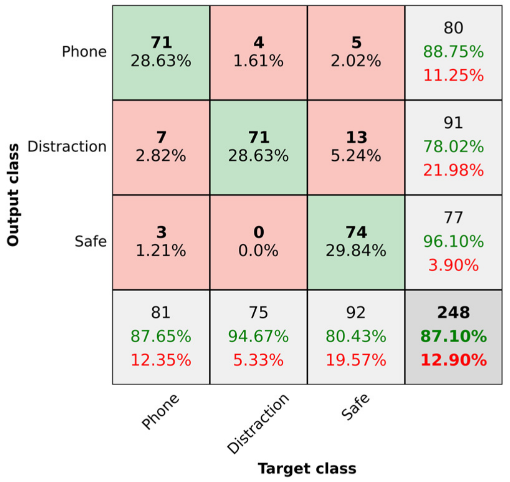
(b)

Figure 6: Second fine-tuned model accuracy (a) and loss (b) for training and validation sets.



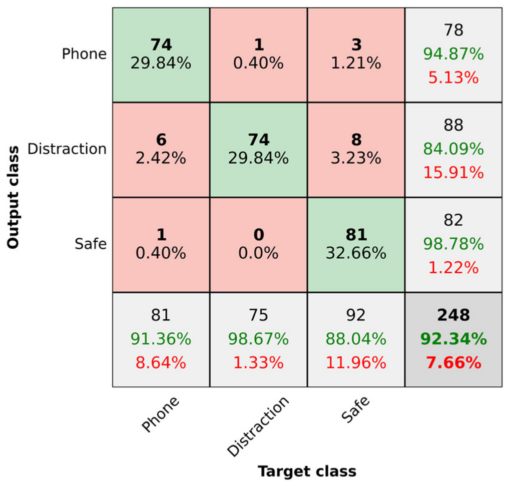
Output class \ Target class	Phone	Distraction	Safe	
Phone	**71** 28.63%	**9** 3.63%	**10** 4.03%	90 78.89% 21.11%
Distraction	**6** 2.42%	**64** 25.81%	**10** 4.03%	80 80.00% 20.00%
Safe	**4** 1.61%	**2** 0.81%	**72** 29.03%	78 92.31% 7.69%
	81 87.65% 12.35%	75 85.33% 14.67%	92 78.26% 21.74%	**248** **83.47%** **16.53%**

Figure 7: Standard model confusion matrix on test set.



Output class \ Target class	Phone	Distraction	Safe	
Phone	**71** 28.63%	**4** 1.61%	**5** 2.02%	80 88.75% 11.25%
Distraction	**7** 2.82%	**71** 28.63%	**13** 5.24%	91 78.02% 21.98%
Safe	**3** 1.21%	**0** 0.0%	**74** 29.84%	77 96.10% 3.90%
	81 87.65% 12.35%	75 94.67% 5.33%	92 80.43% 19.57%	**248** **87.10%** **12.90%**

Figure 8: First fined-tuned model confusion matrix on test set.

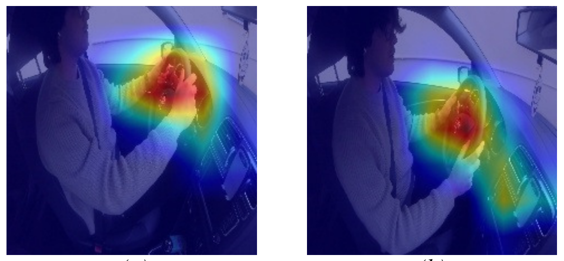


Output class \ Target class	Phone	Distraction	Safe	
Phone	**74** 29.84%	**1** 0.40%	**3** 1.21%	78 94.87% 5.13%
Distraction	**6** 2.42%	**74** 29.84%	**8** 3.23%	88 84.09% 15.91%
Safe	**1** 0.40%	**0** 0.0%	**81** 32.66%	82 98.78% 1.22%
	81 91.36% 8.64%	75 98.67% 1.33%	92 88.04% 11.96%	**248** **92.34%** **7.66%**

Figure 9: Second fined-tuned model confusion matrix on test set.

Table 2: Accuracy and weighted F1-score for each model.

Model	Accuracy	F1-score
Standard	0.8347	0.8352
First fine-tuned	0.8710	0.8716
Second fine-tuned	0.9234	0.9239



(a) *(b)*

Figure 10: The model correctly classifies it as Safe (a). The model misclassifies it as Phone (b).

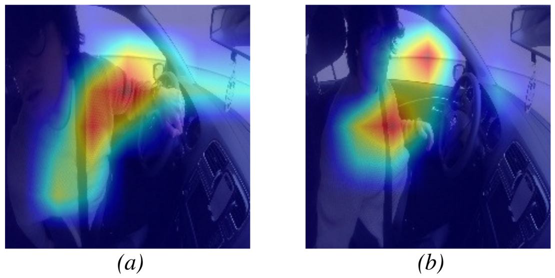


Figure 11: The model correctly classifies it as Distraction (a). The model misclassifies it as Phone (b).

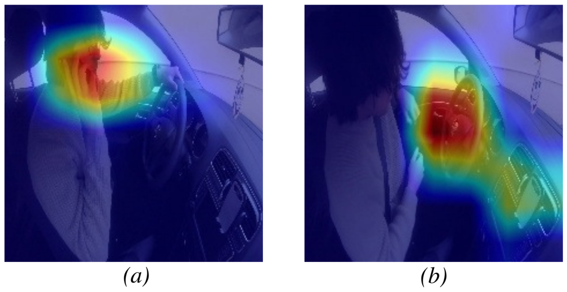


Figure 12: The model correctly classifies it as Phone (a). The model misclassifies it as Distraction (b).

It is worth noting that one frequent cause of misclassification occurs when the driver is moving very fast with respect to the framerate.

6. IMPLEMENTATION ASPECTS

The proposed DAMS require an onboard device capable of executing a neural network based on the second fine-tuned model. For this purpose, a Raspberry Pi 4B [16] has been used as an edge device. It is a compact device with low energy consumption, but it is limited in terms of computing capacity.

Since the architectural model is based on a complex CNN network, it does not achieve an acceptable throughput (in frame per second, *fps*) on this device. To improve its performance, the Raspberry has been equipped with a Neural Compute Stick 2 (NCS2) [17]. NCS2 is a USB stick that provides a dedicated deep neural network hardware accelerator for DNN inference, based on the Intel Movidius Myriad VPU processor.

To run the model on NCS2, it is necessary to install on the Raspberry a toolkit called Openvino [25], for optimizing and deploying deep learning models on Intel devices. Specifically, the Openvino version installed is 2021.4.2. It accepts models according to a format of graph representation and operation set. The graph is represented with XML and binary files, and this representation is referred to as the Intermediate Representation (IR). IR version 10 has been adopted, because the next version is not compatible [21].

The training of the model is carried out on a development server and then deployed on Raspberry for its execution. For this purpose, the Pytorch implementation is converted to the Open Neural Network Exchange (ONNX) format via Python libraries, and then to IR 10.

7. DRIVER'S BEHAVIOR DETECTION PSEUDOCODE

Figure 13 represents in pseudocode the control flow using the classifier for detecting the driver's behavior. Specifically, the configuration parameters of the control flow are the following: (i) the *classificationModel*, set to the second fine-tuned model, as motivated in Section V; (ii) the *fps*, which indicates how many frames per second are captured and processed by the system. The last parameter indicates the frame block dimension used to estimate the driver's behavior. Specifically, a frame block is composed of *N* classified frames. After the acquisition (function *getFrame*), the pre-processing and classification of *N* frames, the system output (variable *Status* in the control flow) is achieved via a majority voting policy for each frame block. In other terms, the most frequently predicted classes will be the block output (classified driver's behavior), as exemplified in Figure 14.

Algorithm 1: Driver's Behaviour algorithm

```
Data: fps, ClassificationModel, N
counterClassified=0; CounterPhone=0;
CounterDistraction=0; CounterSafe=0;
while SystemActive do
    imageRow = getFrame(fps);
    image = preprocessing(imageRow);
    label= frameClassification(ClassificationModel,image);
    counterClassified=counterClassified+1;
    if label==”phone” then
        CounterPhone=CounterPhone+1;
    else
        if label==”Distraction” then
            CounterDistraction=CounterDistraction+1;
        else
            if label==”Safe” then
                CounterSafe=CounterSafe+1;
            end
        end
    end
    if counterClassified==N then
        Status=SelectMax(CounterPhone,
                         CounterDistraction,
                         CounterSafe );
        Activate(Status);
        counterClassified=0; CounterPhone=0;
        CounterDistraction=0; CounterSafe=0;
    end
end
```

Figure 13: Control flow for classifying the driver's behavior via the trained neural network.

Specifically, *N* indicates how many classified frames are needed to determine the overall driver's behavior. This is an important value (also related to the value assigned to *fps)*: a low value would result in many false positives, whereas a high value would capture transient statuses instead of the driver's behavior, thus returning an incorrect classification. In addition, *N* can control the noise due to the fast driver's movements. It is worth noting that, when multiple outputs (e.g., 5) are provided with the same label for both Safe and Phone, the system output is Distraction for safety reasons. For the purpose of testing, the values of *fps* and *N* have been set to 1 and 10, respectively.

Figure 15 shows a pilot test made on 91 seconds of in-car video, via a Pi-Camera, with the related system output, for *fps*=1 and *N*=10. In the end, the system output coincides with the actual driver's behavior.

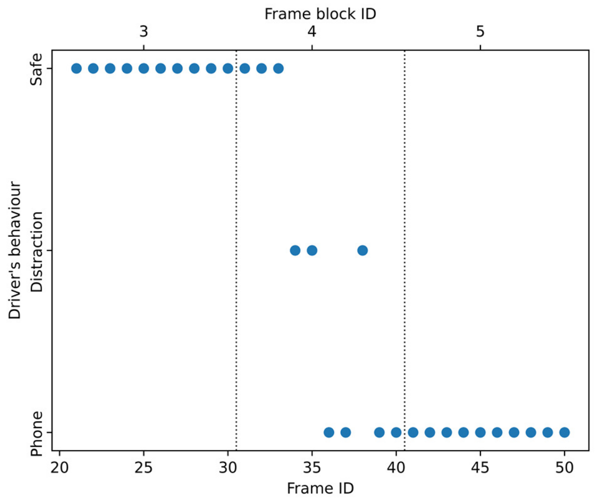


Figure 14: An example of three consecutive frame blocks with fps=1 and N=10 shows that the outputs of the blocks 3, 4, and 5 are Safe, Phone, and Phone, respectively.

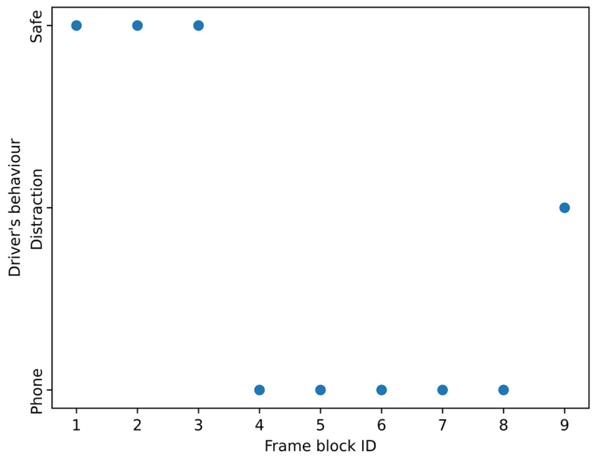


Figure 15: System outputs of in-car video. Note that points 3, 4 and 5 correspond to frame blocks 3, 4 and 5 in Figure 14.

8. PERFORMANCE ANALYSIS OF THE IMPLEMENTATION

This section discusses an overall analysis of the model, deployed on different hardware architectures. Specifically, a Raspberry Pi 4B with NCS2 and NVIDIA Jetson Nano [30] has been used. The performance is measured in terms of time, throughput, and classification accuracy. Timings were recorded on 1,000 images chosen randomly by the training set. Three timings measured in milliseconds have been recorded, varying the set of functions used (FUN): (i) model inference (I); (ii) image pre-processing (PI); (iii) photo capture (CPI). The throughput is computed based on CPI time, and measured in frames per second (FPS). Timings in milliseconds and throughput are summarized in Tab. 3. Here, the best performance is achieved by the Jetson Nano device.

The Openvino model has been evaluated on Raspberry Pi using the testing set and compared to the Pytorch model. According to the results shown in Table 4, the accuracy of the two models is equivalent.

Table 3: Timings (in ms) and FPS for different trials on Raspberry Pi 4B with NCS2, and Jetson Nano.

Device	I	PI	CPI	FPS
Pi with NCS2	144.09	176.97	197.82	5.05
Jetson Nano	63.82	86.77	88.57	11.28

Table 4: Frameworks' performance comparison.

Framework	Accuracy	F1-score
Pytorch	0.9234	0.9240
Openvino	0.9234	0.9240

9. CONCLUSIONS

In this paper, by adopting transfer learning and fine-tuning techniques, a VGG16 neural network has been adapted to recognize three classes of distracting driver's behavior: Safe, Phone (i.e., the driver is using a mobile phone), and Distraction. The best setting has achieved an accuracy of 92.34% on the test set. The best performing model is deployed on a Raspberry 4B edge device, equipped with an NCS2 accelerator, without loss of accuracy. An approach is proposed for detecting driver distraction, on the basis of a majority voting policy. The voting policy has been tested by using sample videos. The throughput in FPS has been evaluated. Specifically, the throughputs achievable on both Raspberry and Jetson Nano boards have been evaluated.

As a future work, the effectiveness of the proposed approach could be evaluated by varying its parameters (*fps* and *N*). Besides, it is worth noting that from the heatmap analysis (and the dataset), it can be said that the developed network does not consider the driver's face to detect the driver's behavior. As a consequence, in case the driver is not focusing on the road, and his hands are on the wheeling, the model will misclassify the driver's behavior. This phenomenon could be investigated as a future work, eventually utilizing a network that also focuses on the driver's gaze, to improve the detection accuracy of driver's behavior.

ACKNOWLEDGEMENTS

Work partially supported by: (i) the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project (Departments of Excellence); (ii) PNRR - M4C2 - Investment 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 1 "Human-centered AI", and Investment 1.5 "Creating and strengthening of "innovation ecosystems", building "territorial R&D leaders", project "THE - Tuscany Health Ecosystem", Spoke 6 "Precision Medicine and Personalized Healthcare", funded by the European Commission under the NextGeneration EU programme; (iii) the Italian Ministry of University and Research (MUR) in the framework of the National Recovery and Resilience Plan, in the National Center for Sustainable Mobility MOST/Spoke10; (iv) the University of Pisa, in the framework of the PRA_2022_101 project "Decision Support Systems for territorial networks for managing ecosystem services"; (v) the Italian Ministry of University and Research (MUR), in the framework of the "Reasoning" project, PRIN 2020 LS Programme, Project number 2493 04-11-2021.

REFERENCES

[1] Volvo Car Corporation, In-car cameras and intervention against intoxication distraction: Animation, https://www.media.volvocars.com/global/en-gb/media/videos/250162/in-car-cameras-and-intervention-against-intoxication-distraction-animation, accessed on April 2023

[2] U.S. Department of Transportation, Distracted Driving Dangers and Statistics, https://www.nhtsa.gov/risky-driving/distracted-driving/.

[3] X. Zhang, et. al. Visual recognition of driver hand-held cell phone use based on hidden crf. In 2011 IEEE Int. Conf. on Vehicular Electronics and Safety, pp 248–251, July 2011.

[4] N. Das, E. Ohn-Bar, and M. M. Trivedi. On performance evaluation of driver hand detection algorithms: Challenges, dataset, and metrics. In 2015 IEEE 18th Int. Conf. on Intelligent Transportation Systems, pp. 2953– 2958, Sept 2015.

[5] T. H. N. Le, et al.. Multiple scale faster-rcnn approach to driver cell-phone usage and hands on steering wheel detection. In 2016 IEEE Conf. on Computer Vision and Pattern Recognition Workshops,.

[6] A. Kashevnik, R. Shchedrin, C. Kaiser and A. Stocker, "Driver Distraction Detection Methods: A Literature Review and Framework," IEEE Access, vol. 9, pp. 60063-60076, 2021.

[7] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection", arXiv:2004.10934, 2020, http://arxiv.org/abs/2004.10934.

[8] W. Liu, D. Anguelov, et al., SSD: Single Shot MultiBox Detector, Cham, Switzerland: Springer, vol. 9905, pp. 21-37, 2015.

[9] S. Ren, K. He, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks", IEEE Trans. Pattern Anal. Mach. Intell., vol. 39(6), 1137-1149, Jun. 2017.

[10] Duy Tran, et al. Real-time detection of distracted driving based on deep learning. IET ITS, Vol. 12(10), pp. 1210-1219.

[11] Ou, C., Ouali, C., Karray, F. . Transfer Learning Based Strategy for Improving Driver Distraction Recognition. In: Image Analysis and Recognition. LNCS, vol 10882. Springer, Cham.

[12] R. Manoharan and S. Chandrakala, "Android OpenCV based effective driver fatigue and distraction monitoring system," 2015 Intern. Conf. on Computing and Communications Technologies (ICCCT), Chennai, India, 2015, pp. 262-266.

[13] Sahoo, G.K., Das, S.K. & Singh, P. A deep learning-based distracted driving detection solution implemented on embedded system. Multimed Tools Appl (2022).

[14] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. 3rd International Conference on Learning Representations (ICLR 2015), 1–14.

[15] Deng, J., Dong, W., et al.. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255).

[16] Raspberry Pi Fundation, Raspberry Pi 4 Model B, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[17] Intel, Intel Neural Compute Stick 2, https://www.intel.com/content/www/us/en/products/sku/140109/intel-neural-compute-stick-2/specifications.html.

[18] H. Park, D. Ahn, et al. "Automatic Identification of Driver's Smartphone Exploiting Common Vehicle-Riding Actions," IEEE Transactions on Mobile Computing, vol. 17(2), pp. 265-278, 2018,

[19] European Road Safety Observatory, Road Safety Thematic Report – Driver distraction, March 2022, available on line at https://road-safety.transport.ec.europa.eu/system/files/2022-04/Road_Safety_Thematic_Report_Driver_Distraction_2022.pdf,

[20] National Center for Statistics and Analysis. (2022, May). Distracted driving 2020 (Research Note. Report No. DOT HS 813 309). National Highway Traffic Safety Administration

[21] Intel, Unable to Use Intermediate Representation (IR) v11 with OpenVINO 2021.3 on Raspberry Pi, https://www.intel.com/content/www/us/en/support/articles/000093146/software.html, accessed on April 2023

[22] Python software fundation, Python, https://www.python.org/.

[23] Linux fundation, Pytorch, https://pytorch.org/

[24] Zhuang, F., Qi, Z., et al. (2020). A comprehensive survey on transfer learning. Proceedings of the IEEE, 109(1), 43-76.

[25] Intel, Openvino documentation, https://docs.openvino.ai/latest/home.html, accessed on April 2023

[26] Ezzouhri, et al (2021). Robust deep learning-based driver distraction detection and classification. IEEE Access, 9, 168080-168092.

[27] R. R. Selvaraju, et al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization, 2017 IEEE Inter. Conf. on Computer Vision Venice, Italy, 2017, pp. 618-626.

[28] Olson, D.L.; Delen, D. Advanced Data Mining Techniques, 1st ed.; Springer: Berlin/Heidelberg, Germany, 2008; p. 138.

[29] De Vitis, G.A.; et al. Fast Blob and Air Line Defects Detection for High Speed Glass Tube Production Lines. J. Imaging 2021, 7, 223. https://doi.org/10.3390/jimaging7110223

[30] NVIDIA, Jetson Nano Developer Kit, https://developer.nvidia.com/embedded/jetson-nano-developer-kit.