DVS4Drivers

Lucrezia Antenucci[1] and Simone Di Rado[1]

[1]Computer Engineering master's degree student, Marche Polytechnic University, Ancona, Italy

Abstract— **Driver drowsiness is considered one of the major causes of motor vehicle accidents on motorways: when driving, the driver is subject to numerous distractions from both outside and inside the vehicle, these distractions negatively affect the driver's ability to react appropriately to anomalous circumstances. In order to mitigate this problem, it was decided to adopt new methods in the field of artificial intelligence such as autonomous driving. In this paper we deal with several methods for the automatic detection of a vehicle driver. Particularly, the main objective is the development of a network that identifies if during the driving the driver has open or closed eyes. We relied on MobileNet v2 and Yolo v5 networks. We choose these two networks because our goal is to evaluate the performance of the networks also on the basis of the type of input data (only the eye or the entire face)**

I. Introduction

One of the main consequences of the health and economic crisis that began in 2020, is the reduction of the mobility and this had a decisive impact on road accidents which decreased by 3% [1]. The leading cause of vehicle accidents is the lack of attention when driving, which is due to external vehicle stimulus, such as advertising billboard and inadequate signage, and internal vehicle stimulus as the use of the smartphone (for instance). Furthermore, another factor to take into consideration is the fatigue behind the wheel maybe for night trips or lack of sleep. All this causes limitations in the ability to react coherently in anomalous circumstances. To circumvent the problem of the distraction while driving, it was decided to adopt the autonomous driving which is to be considered the innovation that represents the future of transport and whose applications will have a significant impact both in the automotive sector and in everyday life.

Recently, a great attention has been devoted to the Deep Learning in view of several improvements in tasks involving text, sound, or image processing. In particular, it has made great strides in artificial vision, such as object detection, motion detection, action recognition and human body positioning estimation. Deep Learning allows computational models of multiple processing layers to learn and represent data with multiple levels of abstraction simulating how the brain perceives and understands multimodal information. Deep Learning is a family of methods that includes neural networks, hierarchical probabilistic models, and a variety of feature learning algorithms both unsupervised and supervised; it represents a Machine Learning technique that use algorithms capable to simulating the human brain. These algorithms are based on the development of neural networking, which represents a software that aims to reproduce the functioning of the human brain.

We point out that Deep Learning find applications in several areas, but the most important one is the automatic driving: thanks to sensors and cameras capable of processing images, automatic driving allows to recognize obstacles in the street, to detect the presence of a pedestrian, to appropriately react to the arrival of an emergency vehicle [2] and in case of danger of activating the braking mechanism. Moreover, the reaction times of an autonomous vehicle are shorter than humans, so it guarantees greater reliability. Hence the autonomous driving will certainly lead to a reduction in accidents. Furthermore, this guidance system would reduce road congestion and it would help in road traffic management. As emphasized in [2], taking advantages of sound recognition algorithms, the vehicle detects the arrival of an emergency vehicle intercepting the siren sound, and exploiting computer vision algorithms the vehicle supervises the behavior of the driver with respect to the emergency vehicle. The main motivation for which there is a great interest from companies in investing in automotive sector concerns not only the development of methods and tools that allow to recognize the degree of fatigue and distraction of the driver, but also to warn him in time if there is a danger by emitting an acoustic signal (for instance).

The purpose of this article is the design and the development of an algorithm for the detection of the fatigue of a driver in real time. To do this we make use of RGB images converted to gray scale, and we implement a neural network to detect the driver's level of attention based on typical human movements that focus on the *eyes, mouth, and head.*

II. State of the Art

As we have already pointed out in the introduction, in recent years the lack of attention to driving is one of the main causes of road accidents [2]. But this problem can be tackled by using different technologies: pattern analysis, eye analysis, facial expression, and physiological signals analysis. The artificial vision could be used to monitor drivers: the face detection, the pose estimation of drivers head and the gaze are key factors to monitor the driver. In this situation the employement of new generation of devices, as event cameras, is fundamental: it allows to acquire high temporal resolution which is a key factor to properly monitor the driver .

III. Methods

We relied on two main methods to classificay the status of an eye (open/closed): the first one is the Gray SCALE, that is the classic conversion from RGB to gray using the opencv library, and the second one is the so-called PCA (principal component analysis). The PCA is a dimensionality reduction method usually employed to reduce the dimensionality of large data sets. In the case we are dealing with this method allows us to achieve better results because a smaller data set is easier to explore. For the actual classification of open eyes or closed eyes we have developed and compared two types of architecture: MobileNet v2; Yolo v5. MobileNet v2 is a convolutional neural network that is much more performing on mobile devices (our case). The MobileNet v2 architecture [3] is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers, and this feature is the opposite of traditional residual models that use expanded representations in the input.

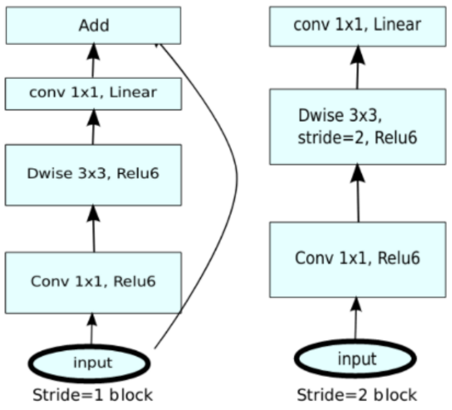


Fig. 1. MobileNet v2 architecture

The MobileNet v2 [4] makes use of slight depthwise convolutions to filter features in the expansion middle layer; moreover the nonlinearities in the narrow layers were removed in order to maintain the representational power. The MobileNet v2 is a classification model developed by google [5]: it provides real-time classifications with computational constraints in mobile devices (as for instance smartphones), and takes advantage of the transfer of learning from ImageNet to the dataset in use. Its architecture has an inverted structure in which the I/O of the residual blocks are bottleneck, i.e. thin bottlenecks. The model also makes use of lightweight convolutions to filter features in the expansion layer, and finally it removes linearity in tight layers.

On the other hand, the Yolo model [6] is a compact and fast object detection model which is very efficient for its size, and is constantly improving. An object detector is designed to generate features from the input images, then uses them through a prediction system to draw bounding boxes around objects in order to predict their classes. The Yolo network consists of three main parts:

- *Backbone*: a convolutional neural network that aggregates and forms image features at different granularities;
- *Neck*: a series of layers to mix and combine the features of the image and then proceed with the forecast;
- *Head*: it reads the features from the neck, retrieves the bounded boxes and proceeds with class prediction.

The are different version of Yolo: the main contribution of Yolo v5 is is to have made innovations in multiple areas of computer vision with the aim of improving the object detection.

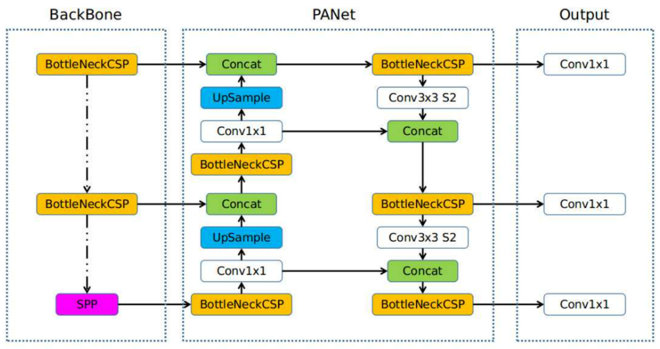


Fig. 2. Yolo v5 architecture

The biggest contribution of Yolo v5 [7] was the transition from the Darknet framework to the PyTorch framework. The functioning of Yolo v5 is the following: for each training batch, Yolo v5 passes the training data to a data loader, which performs an online data augmentation. The types of data augmentation performed by the data loader are the scaling, color space adjustments and mosaic augmentation. Mosaic augmentation helps the model in learning by solving problems related to small objects as the latter are not accurately identified compared to larger ones. Bounding boxes are predicted by Yolo as deviations from a list of anchor box sizes. The anchors in the Yolov5 configuration file are learned automatically based on the training data (all we need to do is to input our data). Yolo v5 formulates the model configuration inside a textit .yaml file, which is condensed to specify only the different layers of the network and then multiplies them by the number of layers in the block. Then it implements the Bottleneck CSP for formulating image features. The CSP solves problems related to the gradient; CSP models are based on DenseNet [8] which was designed to connect layers in convolutional neural networks in order to alleviate the gradient problem (it is difficult to propagate low signals through a deep network), to strengthen feature propagation, encourage the network to reuse features and reduce the number of network parameters.

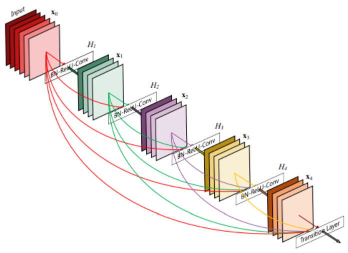


Fig. 3. DenseNet Representation

A. *MobileNetV2 implementation*

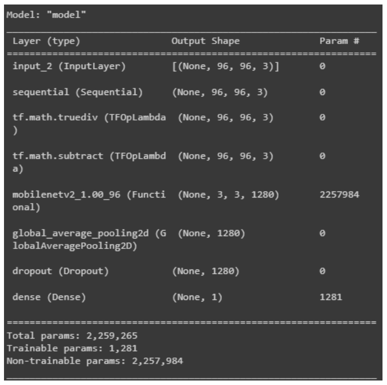
We used the basic model of MobileNet V2 developed by Google: this is pre-trained on the ImageNet dataset, a large dataset consisting of 1.4 million images and 1000 classes. ImageNet is a research training dataset with a wide variety of categories.

First you instantiate a MobileNet V2 model pre-loaded with weights trained on ImageNet. Specifying the include include_top=False argument loads a network that does not include the ratings at the top, which is ideal for extracting features. At this point, the layers of the base created previously using a feature extractor are frozen and a classifier is added to train the first classifier level. It is fundamental that this freeze occurs before training the model as it prevents the weights in a given layer from being updated during training.

When you set layer.trainable = False, the BatchNormalization layer will run in inference mode and will not update the mean and variance statistics. When unlocking a model that contains BatchNormalization levels to perform a Fine Tuning, you need to keep the BatchNormalization levels in inference mode by passing the formation = False when calling the base model. Otherwise, the updates applied to the non-waterproof weights will destroy what the model has learned.

We apply the GlobalAveragePooling2d layer (tf.keras.layers.GlobalAveragePooling2D layer to convert the characteristics of a single vector 1280-element per image) and the dense (we convert the characteristics into a single forecast per image, we don't need an activation function because this prediction will be treated as a logit or raw prediction value. Positive numbers predict class 1, negative numbers predict class 0).

Actual model: we concatenate the data _augmentation, rescaling, base model and feature extractor layers together using the keras api. As mentioned before, we use training = FALSE since our model contains a batch normalization layer. Then we fill in the template before training. Due to the fact that there are two classes, we use binary cross-entropy loss with from _logits = TRUE because the model provides linear output. So we have the following model:



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 96, 96, 3)]       0

 sequential (Sequential)     (None, 96, 96, 3)         0

 tf.math.truediv (TFOpLambda  (None, 96, 96, 3)        0
 )

 tf.math.subtract (TFOpLambd  (None, 96, 96, 3)        0
 a)

 mobilenetv2_1.00_96 (Functi  (None, 3, 3, 1280)       2257984
 onal)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

Fig. 4. MobileNet v2 model

as we can see we have 2.5M frozen parameters, but there are 1002 parameters trailed in the dense state. These are divided into two objects tf.variable objects, the weights and biases. Now we come to the training of the model with 20 epochs, in which we should observe a good accuracy in the validation set (about 94 %). A fine tuning has also been added to the network: one way to further increase the performance of the network is to train (or "fine tune") the weights of the upper layers of the pre-trained model together with the training of the classifier that has been added. The training process will force the weights to be adjusted from generic characteristic maps to characteristics associated specifically with the dataset. You should also try to fine-tune a small number of top layers rather than the entire MobileNet v2 model. In most convolutional networks, the higher a layer is, the more specialized it is. The first few levels learn very simple and generic features that generalize to almost all types of images. Going higher, the characteristics are increasingly specific to the dataset on which the model was trained. The goal of the fine tuning is to adapt these specialized features to work with the new dataset, rather than overriding generic learning. Therefore you have to unlock the base model and set the lower layers to be untrainable. After the fine tuning, the model should reach 98% on the validation set.

B. *Yolo v5 implementation*

The approach followed for the Yolo v5 network is as it follows:

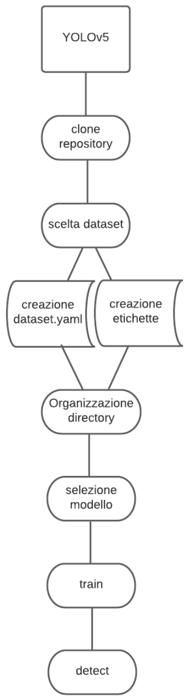


Fig. 5. Yolo v5 steps

- first we cloned the repository on GitHub at the following link `https://github.com/ultralytics/yolov5` and then we installed the necessary dependencies;

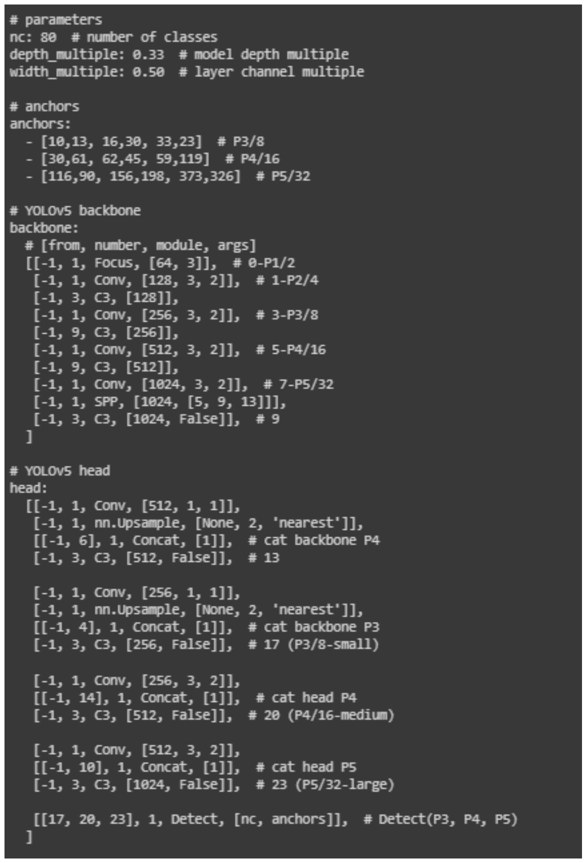
- at this point we have chosen the dataset. The dateset used was the result of the merge between the MRL Eye dataset, the TRIM dataset (set of images downloaded from google through specific queries and subsequently cleaned up) and a dataset composed of frames generated from a video;

- then we configured the *.yaml* file, defining the paths of the folders containing the images used for training and validation, the number of classes (2) and their name ([close, open]);

- after using the *makesense.ai* tool for the image labels, we inserted them in *.txt* format inside a folder that was subsequently imported on the drive to be able to work on them via Google Colab;

- we reorganized our directory so that the network could automatically associate each label with the corresponding image;

- the model we have chosen for the training phase is Yolov5s, which has the following characteristics:



```
# parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, C3, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Fig. 6. Yolov5s model

- at this stage we launch the network training phase on 70% of the dataset (we dedicated the remaining 30% to the validation phase): in this phase we have specified the following parameters: dataset, image size (82), batch -size (16), number of epochs (100);

- finally, we tested the performance of our network, and we observed and evaluated the results obtained.

IV. Experimental protocol

A. Dataset Used

The Dataset used for this project are the following:

1. MRL Eye Dataset [9]: large-scale dataset of human eye images;

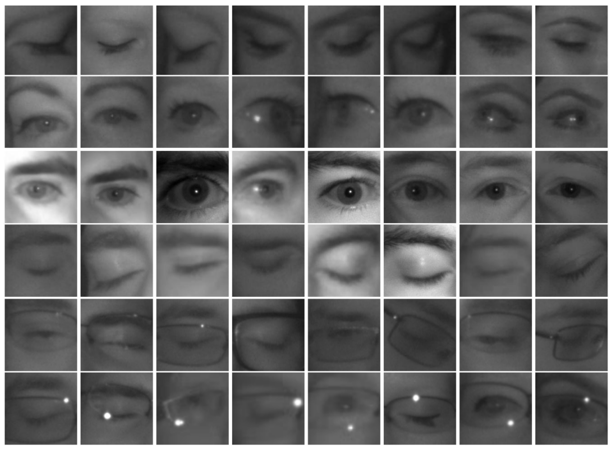


Fig. 7. Example of MRL Eye Dataset

2. Trim Dataset: images downloaded from google using the Serpapi API;

Fig. 8. Example of Trim Dataset

3. VideoFrame Dataset: images generated from a video;

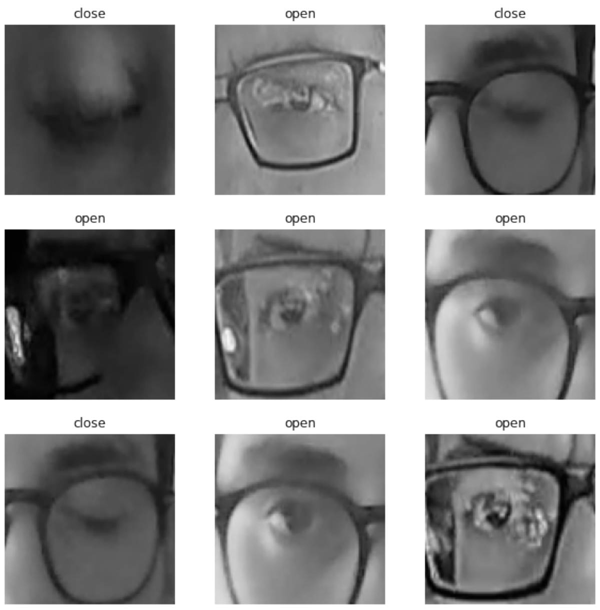


Fig. 9. Example of Video Dataset

4. TestSets: images generated randomly on Google according to appropriate queries and different from the previous ones.

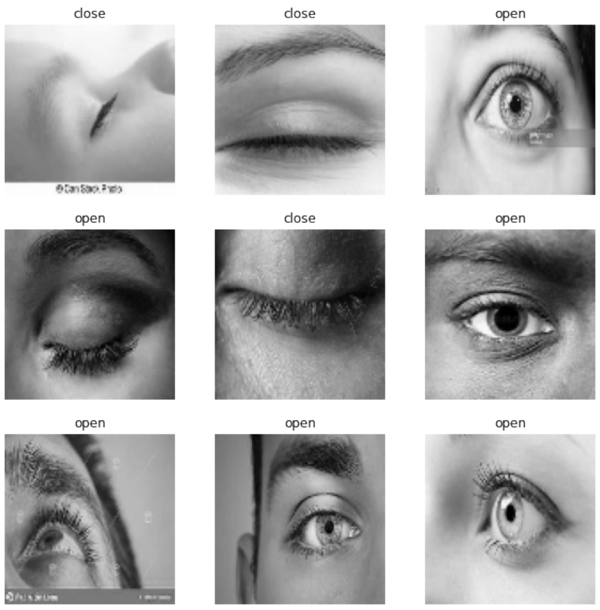


Fig. 10. Example of Test Dataset

B. Dataset Compsition

The first three datasets were used to train the networks by making a merge between them: an estimate of total images is around 150 thousand. We note that the datasets have been generated both in the gray version and in the PCA version. Finally, the latest TestSets dataset was used for the test phase.

V. Results

After the training and development of the networks, we moved on to the analysis of the results. The networks under investigation are the following:

- MobileNet v2 with grayscale conversion of images through the static method;
- MobileNet v2 with grayscale conversion of images using the dynamic method (PCA);
- Yolo v5 with grayscale conversion of images through static method.

A. *MobileNet v2: the static method*

During the pre-training phase we provided a number of initial epochs equal to 20, and a batch-size equal to 64, while the initial loss and accuracy were 1 and 0.49, respectively. With the training phase, after the first 20 epochs, we obtained a loss of 0.14 and an accuracy of approximately 0.94.

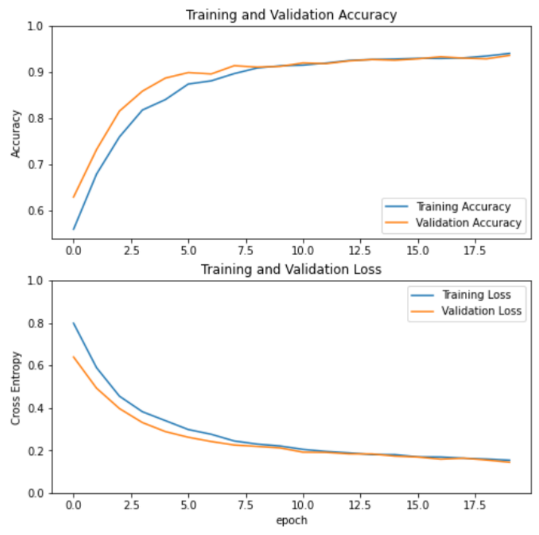


Fig. 11. Training and Validation accuracy, MobileNet v2, static method

Later, again in the training phase, fine tuning was applied and the model trained on another 10 epochs, obtaining a loss of 0.11 and an accuracy of 0.96.

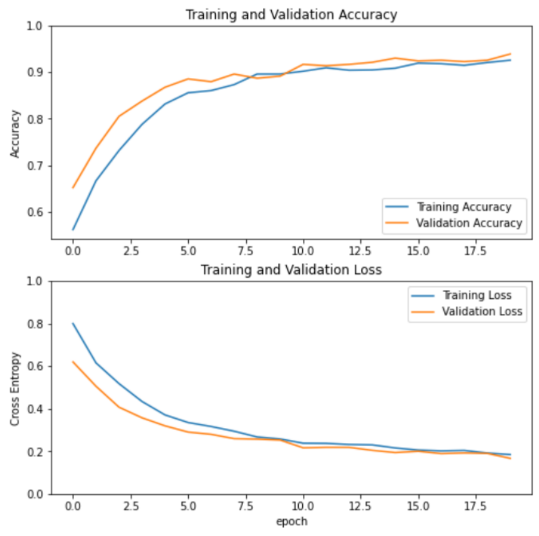


Fig. 13. Training and Validation accuracy, MobileNet v2, dinamic method

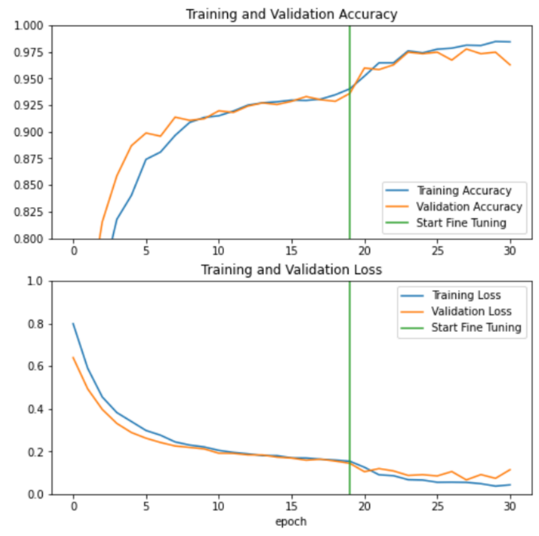


Fig. 12. Training and Validation accuracy, MobileNet v2, static method, Fine Tuning

Finally we tested the network on a new dataset called Test-Datasets (it contains images never seen in the training phase), and we obtain he following results:

- *Loss*: 0.3581;
- *Accuracy*: 0.8700;
- *confusion Matrix*: $\begin{pmatrix} 26 & 3 \\ 1 & 34 \end{pmatrix}$

B. *MobileNet v2: the dynamic method*

As before, we provide a number of initial epochs equal to 20 and a batch-size equal to 64, but with initial loss and accuracy of 0.86 and 0.54, respectively. After the first 20 epochs, we obtained a loss of 0.17 and an accuracy of 0.94. By applying the Fine Tuning we trained the model on another 10 epochs, obtaining a loss of 0.08 and an accuracy of 0.97.

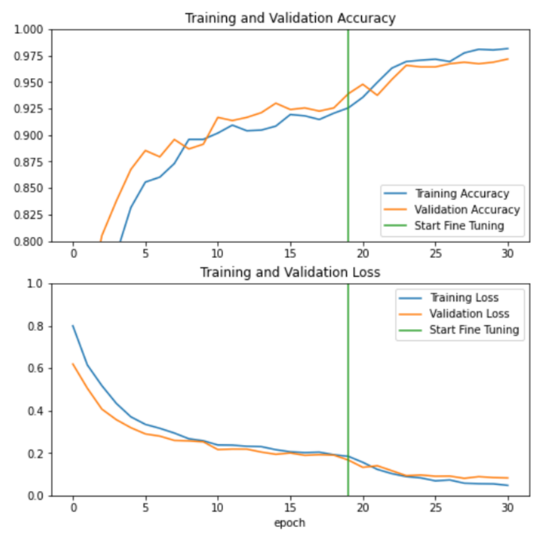


Fig. 14. Training and Validation accuracy, MobileNet v2, dinamic method, Fine Tuning

We tested the network on the same dataset, containing images never seen in the training phase, obtaining the following results:

- *Loss*: 0.3633;
- *Accuracy*: 0.8800;
- *confusion Matrix*: $\begin{pmatrix} 20 & 9 \\ 1 & 34 \end{pmatrix}$

C. *Yolo v5*

In the training phase we set an image-size of 82 and a batch-size of 16, going to perform a total of 50 epochs, and we obtained the following results:

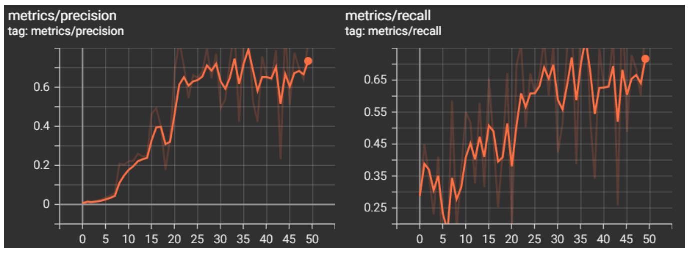


Fig. 15. Precision and Recall, Yolo v5

that is a precision value equal to 0.73 and a recall value equal to 0.72. For what concerns the loss:

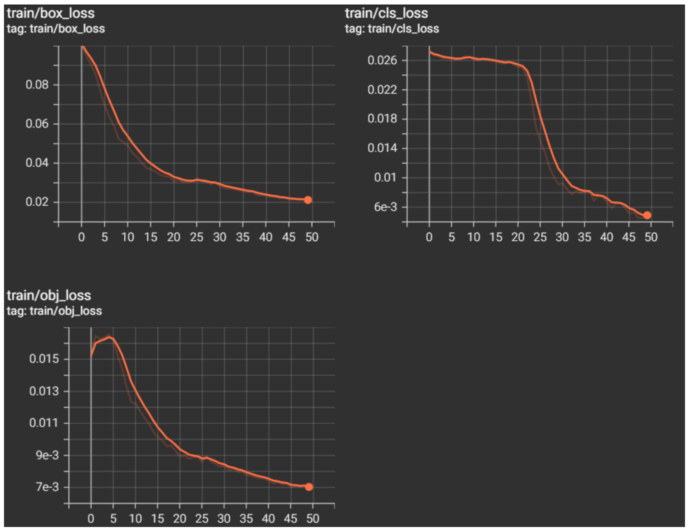


Fig. 16. Loss, Yolo v5

We have a train/box loss of 0.02, a train/cls loss of 4.8459e-3 and finally a train/obj loss of 6.8978e-3. The confusion matrix for the yolo v5 turns out to be the following:

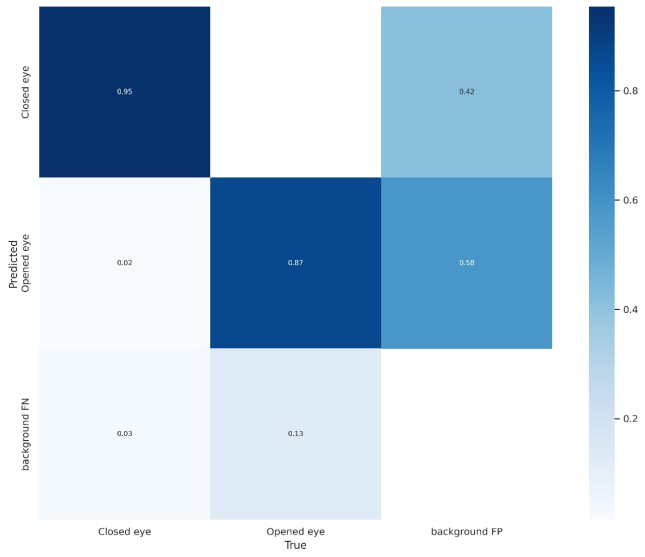


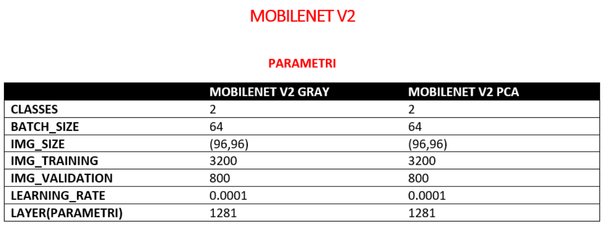
Fig. 17. Confusion Matrix, Yolo v5

D. *Comparison*

Considering that the dataset we used in testing phase has not a considerable number of images, among the three networks under study, we can say, without any doubt, that the worst is the Yolo v5, as it has worse characteristics than the versions of MobileNet v2 both static and dynamic. On the other hand, we can also observe that the MobileNet v2 with the static conversion method and the MobileNet v2 with the dynamic method have very close results. The first seems to be slightly better than the second, but the difference is not relevant in determining which is the best. Based on the results obtained in the test phase and observing the confusion matrix, we believe that the MobileNet v2 with static method is the most performing of the two MobileNet v2: in fact, out of a total of 64 images, the MobileNet v2 with static method commits only 3 errors for the closed eyes class, while the MobileNet v2 with dynamic method commits 9 errors.

VI. Discussion

As we have already observed, a comparison between the three networks under investigation showed that Yolo v5 has the worst characteristics while MobileNet v2 with the static method and with the dynamic method present results that are close to each other. Let's summarize this in a comparison summary table:

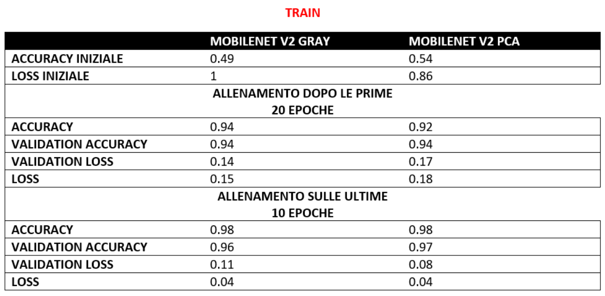


MOBILENET V2

PARAMETRI

	MOBILENET V2 GRAY	MOBILENET V2 PCA
CLASSES	2	2
BATCH_SIZE	64	64
IMG_SIZE	(96,96)	(96,96)
IMG_TRAINING	3200	3200
IMG_VALIDATION	800	800
LEARNING_RATE	0.0001	0.0001
LAYER(PARAMETRI)	1281	1281

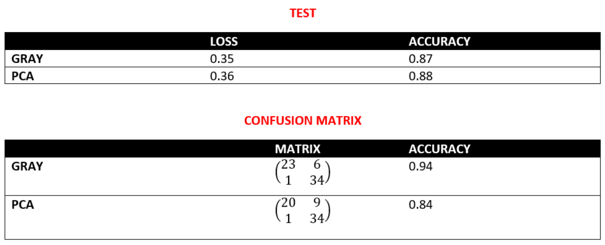
Fig. 18. Tabella dei parametri



TRAIN

	MOBILENET V2 GRAY	MOBILENET V2 PCA
ACCURACY INIZIALE	0.49	0.54
LOSS INIZIALE	1	0.86
ALLENAMENTO DOPO LE PRIME 20 EPOCHE		
ACCURACY	0.94	0.92
VALIDATION ACCURACY	0.94	0.94
VALIDATION LOSS	0.14	0.17
LOSS	0.15	0.18
ALLENAMENTO SULLE ULTIME 10 EPOCHE		
ACCURACY	0.98	0.98
VALIDATION ACCURACY	0.96	0.97
VALIDATION LOSS	0.11	0.08
LOSS	0.04	0.04

Fig. 19. Tabella dei risultati del train



TEST

	LOSS	ACCURACY
GRAY	0.35	0.87
PCA	0.36	0.88

CONFUSION MATRIX

	MATRIX	ACCURACY
GRAY	$\begin{pmatrix}23 & 6\\ 1 & 34\end{pmatrix}$	0.94
PCA	$\begin{pmatrix}20 & 9\\ 1 & 34\end{pmatrix}$	0.84

Fig. 20. Tabella dei risultati del test

The MobileNet v2 with static conversion method seems to be slightly better than the second in terms of loss, while the second is more efficient in terms of accuracy. Since the difference between the two, in terms of performance, is not relevant in order to establish which is the best, based on the results obtained in the test phase and observing the confusion matrix, we believe that the MobileNet v2 with static conversion method of the images is the most performing. In fact, out of a total of 64 images, it only commits 3 errors for the closed eyes class, compared to 9 errors committed by the MobileNet v2 network with conversion using the dynamic method.

VII. Conclusions

We conclude this study considering that the results obtained for the two versions of MobileNet v2 seem very satisfactory to us. The chosen network offers excellent performance and could be of great contribution to solving the problem of driving distraction. Implemented within a mobile device, this solution would be of great help for autonomous driving and would have not only an important but also revolutionary impact within the automotive sector. In this project we focus our attention on the driver's gaze, and in particular on closing and opening the eyes and on the direction of the gaze (project related to ours, but carried out by another group). The next goal could be to develop a network that takes care of detecting the driver's face, estimating the position of the head and evaluating the movements of the body. Another challenge could the improvement of performance to enable real-time processing evaluating also the robustness to changes in light, cost and privacy.
The first step to proceed with the project could be to test the network on an infrared video, extrapolating the frames from the video and using them as input images for the network: in this way we can see if the performance varies and how much the performance varies, if there is a performance degradation or if the network continues to perform well even in this case.

References

[1] "Incidenti stradali". In: (2020).

[2] M. Cantarini, L. Gabrielli, L. Migliorelli, A. Mancini, and S. Squartini. "Beware the Sirens: Prototyping an Emergency Vehicle Detection System for Smart Cars". In: (2022).

[3] `https://mmclassification.readthedocs.io/en/latest/papers/mobilenet_v2.html`.

[4] *https://pytorch.org/hub/pytorch*$_v$*ision*$_m$*obilenet*$_v$*2/*.

[5] *https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html1*.

[6] *https://blog.roboflow.com/yolov5-improvements-and-evaluation/*.

[7] Glenn Jocher, Alex Stoken, Jirka Borovec, Liu Changyu, Adam Hogan, L Diaconu, F Ingham, J Poznanski, J Fang, L Yu, et al. "ultralytics/yolov5: v3. 1-bug fixes and performance improvements". In: *Version v3* 1 (2020).

[8] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: (2022).

[9] `http://mrl.cs.vsb.cz/eyedataset`.