Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Distracted Driver Detection using Deep Learning Techniques

Authors:
Alessandro Riboni - 847160 - a.riboni2@campus.unimib.it
Davide Sangalli - 848013 - d.sangalli5@campus.unimib.it
Federico Signoretta - 847343 - f.signoretta@campus.unimib.it

May 11, 2020

Abstract

Every year, millions of people are involved in car accidents due to distracted driving. This report explains how it is possible to prevent this problem by exploiting Deep Learning techniques for an images classification task, mainly applying Transfer Learning through the Fine-Tuning method. Several architectures are compared: a basic CNN from scratch and some architectures based on a pre-trained model such as MobileNet, VGG16 and VGG19. The use of imagenet weights as initial ones is fundamental to develop a high-performance model. Moreover, to adapt the model to the problem a Sequential Model-Based Optimization is performed, allowing to achieve satisfying results on new images, demonstrating a high level of generalization.

1 Introduction

Every year, According to the World Health Organization, millions of people die (1.35 million) or suffer non-fatal injuries, with many incurring a disability (between 20 and 50 million) due to road traffic crashes. The main risk factors can be summarized in human error, speeding, driving under the influence of alcohol and other psychoactive substances, non-use of motorcycle helmets, seat-belts, and child restraints, unsafe road infrastructures and vehicles, inadequate post-crash care and law enforcement of traffic laws and, finally, distracted driving. In particular, "distracted driving is any action that diverts attention from driving, including talking or texting on your phone, eating and drinking, talking to people in your vehicle, fiddling with the stereo, entertainment or navigation system - anything that takes your attention away from the task of safe driving"[1].
Each of these actions could be very dangerous to oneself and others, so it is essential to detect these harmful behaviours and act appropriately. The main idea of this project is to build a model able to recognize the driver actions to reduce the number of accidents. How is it possible to reach this aim? From a practical point of view, it is necessary to install a little camera inside cars and recording the images or process them in real-time (it depends on the aim). The video is sampled and it returns a certain number of frames. At this point, the images are ready to be processed by the models, which classify the driver's action.
Image classification is a tricky task, but exploiting the huge potential of the Deep Learning techniques, it is possible to create very powerful classifiers based on Neural Networks. In particular, the class of deep neural networks capable of reaching satisfying performance on this type of task is Convolutional Neural Network (CNN). CNNs are biologically-inspired architectures made up of neural networks stuff and they are the most used architectures in computer vision. Convolutional layers convolve the input and pass its result to the next layer.
One of the most important features of CNN is weights sharing: the vector of weights and the bias are called filters and represent particular features of the input (e.g., a

particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting[2].
Often, in order to solve this kind of task and optimizing space in memory and computational time, transfer learning technique is used. The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. The new models can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.
Through this type of algorithms, it is possible to associate an image to a certain class (in this case the driver action) with a percentage of accuracy of belonging to that class.
At this point, the last fundamental question is: "how can this classification be useful?". The answer depends on the final scope. From an insurance point of view, it can be useful as automatic evidence of car crash: the model provides automatically if the driver was distracted or not - through the recorded images before the crash - specifying which kind of action he was doing. Moreover, it can be useful as a driver's behaviour indicator: if the driver drove most of the time safely, then he gets a discount on the next insurance instalment. In this way, it could raise awareness of the driver's behaviour, with a consequent decrease in car crashes. Finally, it can be used in the same way as the seat belt signals are used: overpassed a certain speed (e.g. 15 km/h), the images are processed in real-time, providing a classification of the action and, if the driving is not safe, the car activates an acoustic signal.

2 Datasets

The data are taken from a Kaggle challenge published on *5th April 2016* by State Farm and are composed by one CSV file, a directory containing the training images and a directory containing the test images[3].
The CSV file is referred to the training images and covers a crucial role: it allows the development of k-fold cross-validation based on drivers. It is composed of three columns: the name of the image called *img*, the driver's identifier called *subject* and the driver's action called *classname*.
The driver's actions are 10, labelled as *c0,...,c9* and described in the table 1. The training directory is composed by 10 sub-directories, called as exactly the above labels. Inside each sub-directory the images associated with that class are contained, as shown in figure 1. The total number of images in the training directory is $22,424$. In general, the classes are balanced, with an average of about $2,242$ images per class. An exception is class *c8*, with less than $2k$ images. Moreover, the number of images per class is balanced for each driver as well. The total number of different drivers is

26. The infographic in the appendix A, figure 8, shows the count of images per class and driver.

class	**action**
c0	Safe driving
c1	Texting - right
c2	Talking on the phone - right
c3	Texting - left
c4	Talking on the phone - left
c5	Operating the radio
c6	Drinking
c7	Reaching behind
c8	Hair and makeup
c9	Talking to passenger

Table 1: Description of drivers' actions with its respective labels

Finally, the test directory is composed of 79,726 non-labelled images. Even if the test images have no labels, they play an essential role in the final stage of the performance verification. In facts, it is possible to obtain a score performance (measured by log loss) directly from the *Kaggle* challenge.

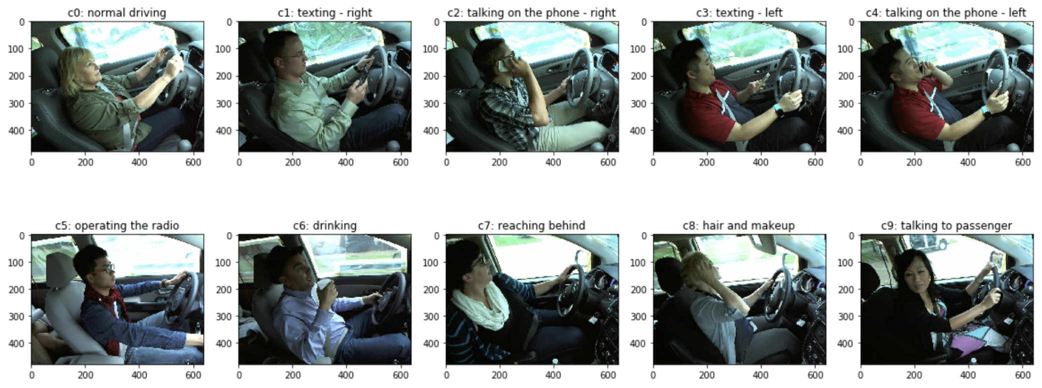

Figure 1: Driver actions

3 The Methodological Approach

This section describes the methodological approach used to develop the project. It explains the Image Data Augmentation process performed on the training images, then presents the cross-validation method used to compare different architectures. Finally, there is a description of the Bayesian Optimization process performed on the hyper-parameters of the model chosen to predict the images of the test set. These phases are implemented in `Python 3.7` using mainly `TensorFlow`, `Scikit-learn`, `Numpy`, `OpenCV` and `Optuna`. In the following sections class and methods names belonging to these libraries will be used.

3.1 Image Data Augmentation

Image Data Augmentation is a technique used to create new training images from existing ones. This method is often used to increase the number of data to develop more robust models. However, it is also helpful to make the classifier more flexible to generalize better on new images. The presence of 26 different drivers within the dataset has made it necessary to use this technique mainly to limit the learning of the physical characteristics of the various drivers and to reduce overfitting on images of new drivers not presented during the training phase.
This process is done through the `ImageDataGenerator` class of the `Keras` library. This iterator returns a *batch* composed of 32 augmented images, which are obtained through random *height* and *width shifts*, *rotated* within a range of 20 degrees and *zoomed* to try to identify the actions of the drivers.
This technique has been applied only to images used to train networks. Instead, all data were reshaped, scaling the images to 128×128 pixels, normalizing the pixels values within the range `[0,1]`, but keeping the three colour channels.

3.2 K-fold cross-validation

One of the elements that make this task challenging and ambitious is the management of the drivers within the training, validation and test set. The images provided by State Farm are frames from videos and a random division in training and validation could lead to a not very robust model due to the presence of very similar frames where the same subject performs the same action. Also, the objective is to apply the model developed on new drivers so that it should classify their actions correctly even if they have not used their images during the training phase.
For these reasons, stratified cross-validation is performed for drivers with $k = 5$. In this way, each driver is present once in the validation set and four times in the training set. The choice of the number of *k-folds* is made to keep similar the ratio between the number of training and validation images and for computational reasons as well. Not having the same number of images per driver, the ratio is not the same for each fold. For this reason, during the architectures comparison and optimization phase, the chosen metric is multiplied by the number of images in the validation set.

3.3 Description of architectures

In order to solve this image classification task, a CNN model from scratch and some pre-trained networks are compared. In particular, fine-tuning techniques are applied to the following pre-trained networks: `VGG16`, `VGG19` and `MobileNet`. Before diving deeper into the architectures, some details are worth to be mentioned. First of all, the pre-trained networks are downloaded using the `imagenet` weights, with `include_top=False`, that means they have no default fully-connected layers.

3.3.1 Definition of extra_layers

To make objective comparisons and to adapt the nets to the problem, some extra layers are added at the end of the considered networks. In particular, these layers, called `extra_layers`, are:

- a `Flatten` layer;
- a `Dense` layer with 128 neurons and a `Dropout` layer with 0.3 `Dropout` rate;
- a `Dense` layer with 64 neurons and a `Dropout` layer with 0.1 `Dropout` rate.

All the `Dense`s have a *relu* activation function. Finally, there is an output layer of 10 neurons corresponding to the 10 output classes, with a *softmax* activation function.

3.3.2 CNN from scratch

The first model proposed is a CNN implemented from scratch, called `CNN_scratch`. Its architecture consists of four blocks and `extra_layers` as final block, implemented as follows:

1. first block: two `Conv2D` layers of 16 neurons each, with `relu` activation function, `kernel_size=3`, and `padding='same'`. In addiction, a `BatchNormalization` layer and a `MaxPooling` layer of `size=2`;
2. second block: two `Conv2D` layers of 32 neurons each, with `relu` activation function, `kernel_size=3`, and `padding='same'`. In addiction, a `BatchNormalization` layer and a `MaxPooling` layer of `size=2`;
3. third block: three `Conv2D` layers of 64 neurons each, with `relu` activation function, `kernel_size=3`, and `padding='same'`. In addiction, a `BatchNormalization` layer and a `MaxPooling` layer of `size=2`;
4. fourth block: three `Conv2D` layers of 128 neurons each, with `relu` activation function, `kernel_size=3`, and `padding='same'`. In addiction, a `BatchNormalization` layer and a `MaxPooling` layer of `size=2`.
5. final block: `extra_layers`

3.3.3 VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition"[4]. This network is composed of 5 convolutional blocks and fine-tuning is performed at two different levels, called `VGG_16_trainable_1` and `VGG_16_trainable_2`.
`VGG_16_trainable_1` refers to the architecture where the last two blocks and the `extra_layers` (defined in subsection 3.3.1) are trained together, while the first three blocks have the Imagenet weights. In this model, the total number of parameters is about 15.8 million, of which about 8.1 million trainable.
Instead, `VGG_16_trainable_2` refers to the architecture where the last block and the `extra_layers` (defined in subsection 3.3.1) are trained together, while the first four blocks have the Imagenet weights. In this model, the total number of parameters is about 15.8 million, of which about 14 million trainable.
The choices are made considering a trade-off between computational issues and the exigency of having to re-train part of the network to adapt it better to the problem.

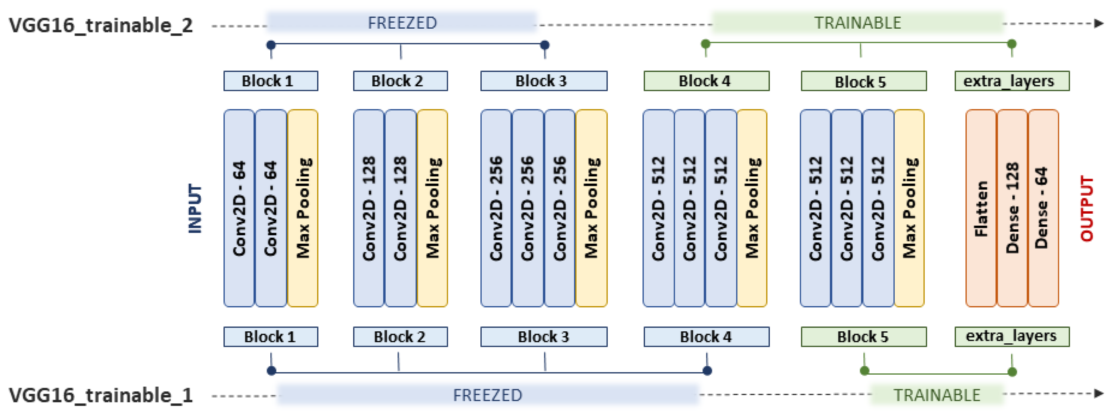

Figure 2: Graphical representation of `VGG_16_trainable_2` and `VGG_16_trainable_1`

3.3.4 VGG19

VGG19 is conceptually similar to VGG16 but has an additional layer in the last three convolutional blocks. This network is composed of 5 convolutional blocks and fine-tuning is performed at two different levels, called `VGG_19_trainable_1` and `VGG_19_trainable_2`. The `VGG_19_trainable_1` refers to the architecture where the last two blocks and the `extra_layers` (defined in subsection 3.3.1) are trained together, while the first three blocks have the Imagenet weights. In this model, the total number of parameters is about 21.1 million, of which about 10.5 million trainable.

Instead, `VGG_19_trainable_2` refers to the architecture where the last block and the `extra_layers` (defined in subsection 3.3.1) are trained together, while the first four blocks have the Imagenet weights. In this model, the total number of parameters is about 21.1 million, of which about 18.8 million trainable.
The choices are made considering a trade-off between computational issues and the exigency of having to re-train part of the network to adapt it better to the problem.

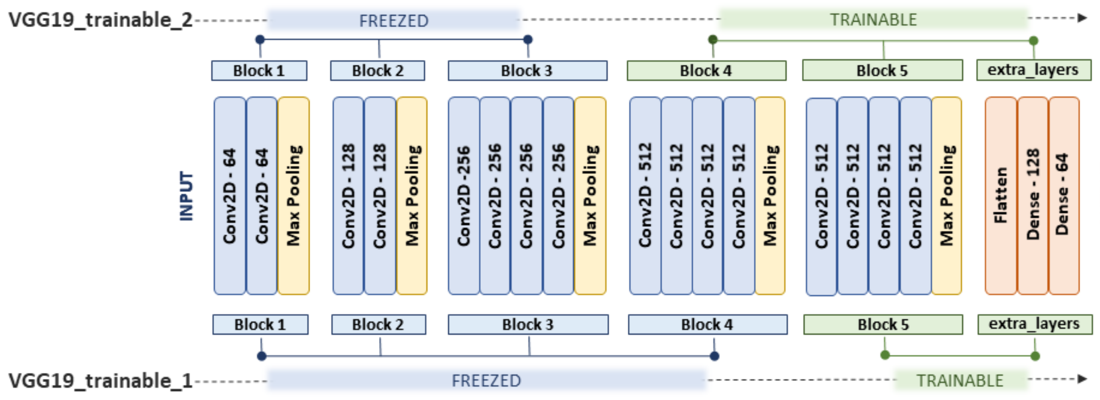

Figure 3: Graphical representation of `VGG_19_trainable_2` and `VGG_19_trainable_1`

3.3.5 MobileNet

MobileNet is a model for mobile and embedded vision applications. It is based on a streamlined architecture that uses depth-wise separable convolutions to build lightweight deep neural networks. In this way, it is possible to deal with a trade-off between latency and accuracy[5]. Due to the lower complexity, the network is trained entirely. The total of the parameters, considering the extra layers as well, is about 5.3 million. This model is called `MobileNet_trainable_all`.

3.3.6 Configuration of the learning process

All the considered models are compiled using *categorical crossentropy* as loss function, *accuracy* as metric, *Stochastic Gradient Descent* as optimizer, with default parameters (`learning_rate=0.01` and `momentum=0.9`) and number of epochs equal to 20. In particular, after several trials, the choice of SGD was dictated by greater stability than Adam and RMSprop. During the training phase, `EarlyStopping` and `ModelCheckPoint` are used. The former is useful to interrupt the training process when the `validation_loss` stops improving. More specifically, patience parameter is set to 5, meaning that if validation loss does not improve within five epochs, the training process stops. The latter is useful to save the model configuration, which minimized the `validation_loss`.

The performance of the models are measured through *log loss* because it takes into account the uncertainty of predictions based on how much it varies from the actual label. As above mentioned, using the 5-folds cross-validation, it is necessary to evaluate the weighted average for the number of images for each fold. In addition to comparisons based on log loss, model *accuracy* metrics are also considered.
The configuration of the `extra_layers` and the learning process of the best model will be optimized through an SMBO process.

3.4 Optimization

After comparing the different architectures, it is decided to optimize the hyper-parameters of the best model: `MobileNet_trainable_all`. In particular, the Sequential Model-Based Optimization (SMBO) is carried out, which minimizes the validation loss by selecting at each iteration a set of hyper-parameters through bayesian reasoning. Differently to the uninformed search, typical of Grid and Random Search, the selection is performed taking into account the results of previous executions. This process is performed through Optuna, an open-source hyperparameter optimization framework to automate the hyperparameter search [6]. It allows optimizing TensorFlow hyper-parameters by defining a wrap model with an objective function to be minimized, suggesting the search space of hyper-parameters using a trial object and, finally, creating a study object that allows performing the optimization. The weighted average of the log loss, obtained in the k-iterations of the model, is chosen as the objective function to be minimized. The optimization process concerns both the structure of `extra_layers` and the parameters of SGD optimizer. The choice of hyper-parameters and the related search spaces are:

- `activation`: activation function of both `Dense` layers `[`*`"relu","leakyRelu"`*`]`;
- `n_units_1`: number of neurons of first `Dense` layer, range of integer `[64, 512]`;
- `dropout_rate_1`: rate of first `Dropout` layer, range of discrete parameters `[0, 0.5]` with a step of discretization equal to 0.1;
- `n_units_2`: number of neurons of second `Dense` layer, range of integer `[32, 256]`;
- `dropout_rate_2`: rate of second `Dropout` layer, range of discrete parameters `[0, 0.5]` with a step of discretization equal to 0.1;
- `learning_rate`: value of learning rate of SGD, range `[1e-4, 1e-2]` in the log domain;
- `momentum`: value of momentum of SGD, range `[0.5, 0.9]` in the log domain.

Furthermore, two essential features of this framework have been exploited, namely the possibility to use a *Pruner* and a *Sampler*. The Pruning process makes it possible

to systematically reduce the research space by evaluating the value of the objective function already in the early epochs. Due to a large number of images and having to fit the model on the 5-folds, a function has been defined that at each trial would clear RAM or GPU made available by Google Colab. To exploit this function even in trials to be pruned, it is necessary to override the `KerasPruningCallback` class to postpone the "raise" of the `TrialPruned()` exception at the end of the fitting on the first fold. This adjustment led to a slowdown but made it possible to complete the optimization process with 50 trials using a limited amount of RAM. The Sampler, on the other side, is fundamental to sample the search space of the parameters. The TPE sampler was used, which applies the Tree-structured Parzen Estimator algorithm. As reported in Optuna's documentation "on each trial, for each parameter, TPE fits one Gaussian Mixture Model (GMM) $l(x)$ to the set of parameter values associated with the best objective values, and another GMM $g(x)$ to the remaining parameter values. It chooses the parameter value x that maximizes the ratio $\frac{l(x)}{g(x)}$" [6].
After evaluating the kind of search space parameters (both continuous and discrete) and performing several tests, it is decided to use the Gaussian Process (GP) as a surrogate model and the Lower Confidence Bound (LCB) as acquisition function. The optimization is performed on 50 trials using a database to save all the information obtained and to keep track of the history of the trials done. Below there is the best configuration obtained at trial 15:

hyper-parameter	**value**
`n_units_1`	387
`dropout_rate_1`	0.1
`n_units_2`	107
`dropout_rate_2`	0.2
`activation`	*relu*
`learning_rate`	0.008136
`momentum`	0.51889

This configuration improved the performance of the `MobileNet_trainable_all` obtained when comparing architectures. For this reason, it is decided to use this model to predict the test images.

3.5 Prediction of test images

In order to assess the model's capability, through the Kaggle platform, predictions of the test set images are performed. The prediction is made through the average of the predictions obtained by the 5-folds. This process has allowed to obtain a model that better generalizes on the new images. Finally, a small demo is built on a sample of two new drivers' images to display the results obtained.

4 Results and Evaluation

4.1 Selection of the best model

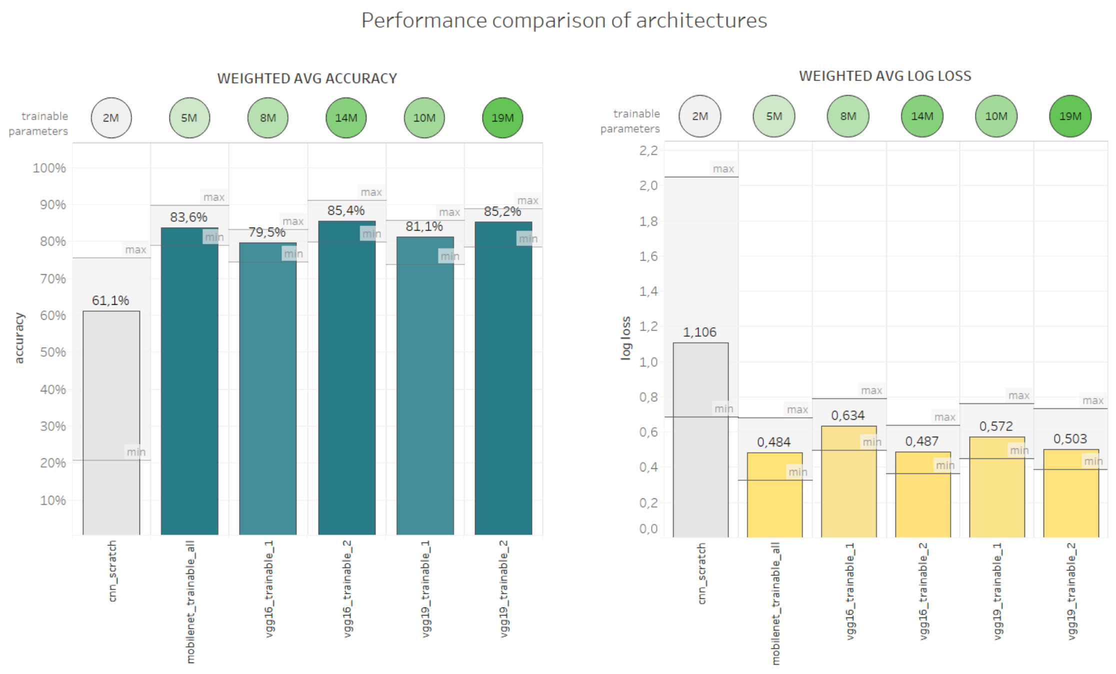

Figure 4: Comparison of different architectures. The weighted accuracy and log loss on 5-validations for each architecture are reported.

4.2 Optimization of the best model

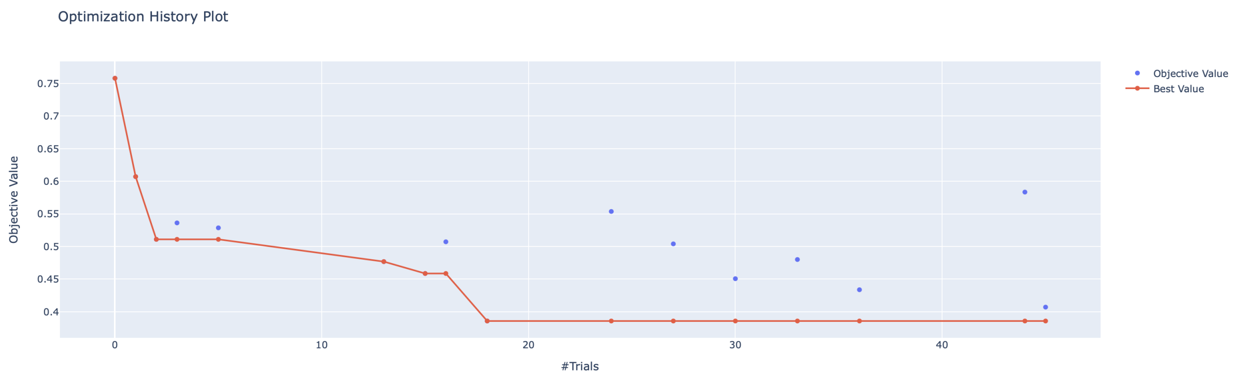

Figure 5: History plot of the optimization process.

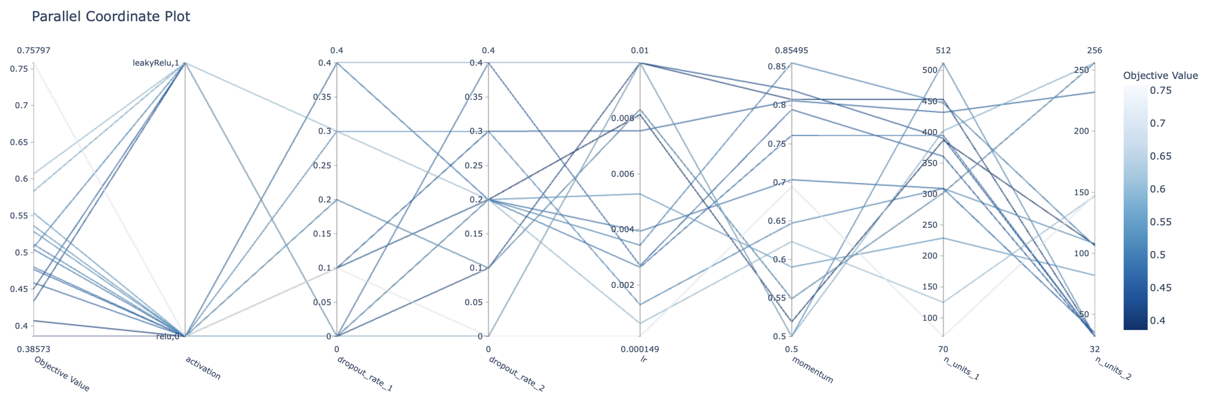

Figure 6: Parallel plot of the completed trials.

4.3 Results of the best configuration

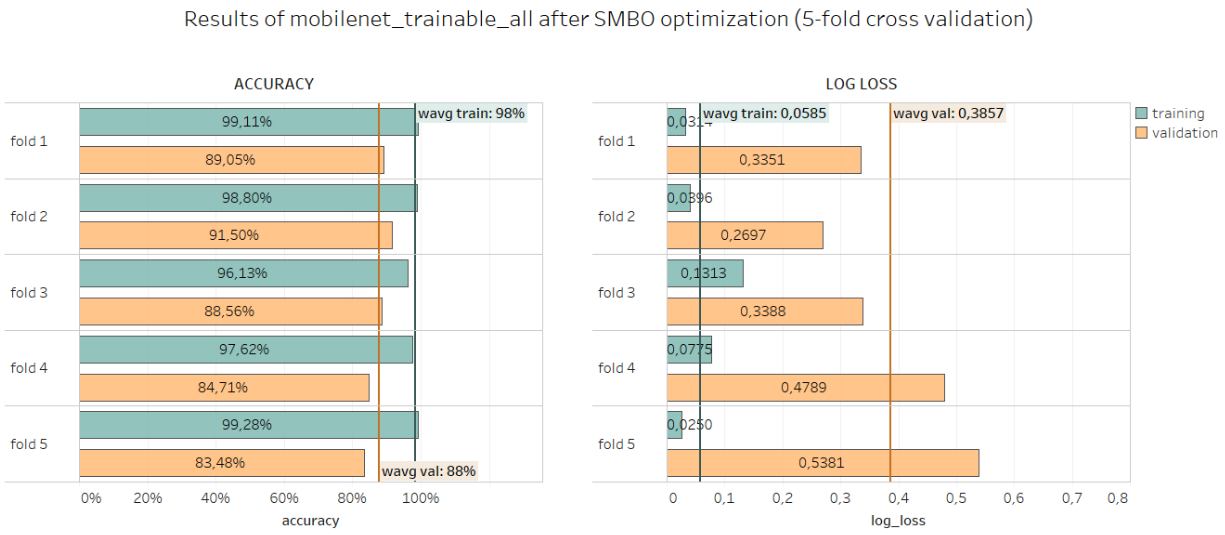

Figure 7: 5-folds comparison of `mobilenet_trainable_all` with the optimized configuration.

	log loss
Training images	0.0585
Validation images	0.3857
Test images	0.3489

Table 2: log loss values on training (weighted average), validation (weighted average) and test images.

5 Discussion

The choice of the best model is made by evaluating the *log loss* values. In particular, it is fundamental to start from the `imagenet` weights, as can be seen from the charts of the pre-trained architectures compared to `cnn_scratch`. Moreover, training more final layers of the networks allows to fit them better to the problem and to obtain higher performance.
The best models are `VGG16_trainable_2` and `MobileNet_trainable_all`, as shown in Figure 4, even if the performances are basically on the same level. Due to the smaller number of trainable parameters (about $5M$), it is decided to optimize the configuration of the `MobileNet_trainable_all` model. The choice of a model with a reduced number of parameters is essential to process images in real-time on limited hardware installed on cars.
From the images presented in subsection 4.2, it is possible to observe the improvement of the objective function during the SMBO process. By comparing the results with the initial parameters, the smaller `learning_rate` and `momentum` values, the higher number of neurons in both layers and the low dropout values allow an improvement in terms of log loss. An increased number of neurons in the fully-connected layers involves an increase in the number of trainable parameters (about $9.6M$).
With the optimized configuration, the average log loss value on the validation set is 0.3857, as shown in Figure 7. The model's generalization capability can be considered satisfying; in fact, a log loss score of 0.3489 on more than $79k$ images of the test set is obtained.

6 Conclusions

The work undertaken has made it possible to develop a model for the classification of actions carried out by a driver. The results obtained are satisfying. The most difficult categories to classify are *safe driving* (*c0*), *hair and makeup* (*c8*) and *talking to passengers* (*c9*). One of the reasons is undoubtedly the similarity between the images of these three classes. Another one could be the ambiguous labelling of some frames within the training set. Checking the labels of these classes and increasing the c8 class images could lead to an improvement in the classifier.
Finally, a possible future work could be to focus on the relationship/connection between the complexity of the proposed network and the type of hardware used to acquire the images. One of the possible approaches could be the ensemble of several pre-trained models in order to improve the overall performance. This approach is computationally expensive, compromising its use on reduced hardware.

References

[1] NHTSA, “Distracted driving,” in *https://www.nhtsa.gov/risky-driving/distracted-driving*, 2018.

[2] LeCun and Yann, “Lenet-5, convolutional neural networks,” 2013.

[3] State-Farm, “Distracted driver detection,” in *https://www.kaggle.com/c/state-farm-distracted-driver-detection/data*, 2016.

[4] K. Simonyan and A. Zisserman, “Very Deep Convolutional Network for Large-Scale Image Recognition,” 2015.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017.

[6] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Appendix A Infographic

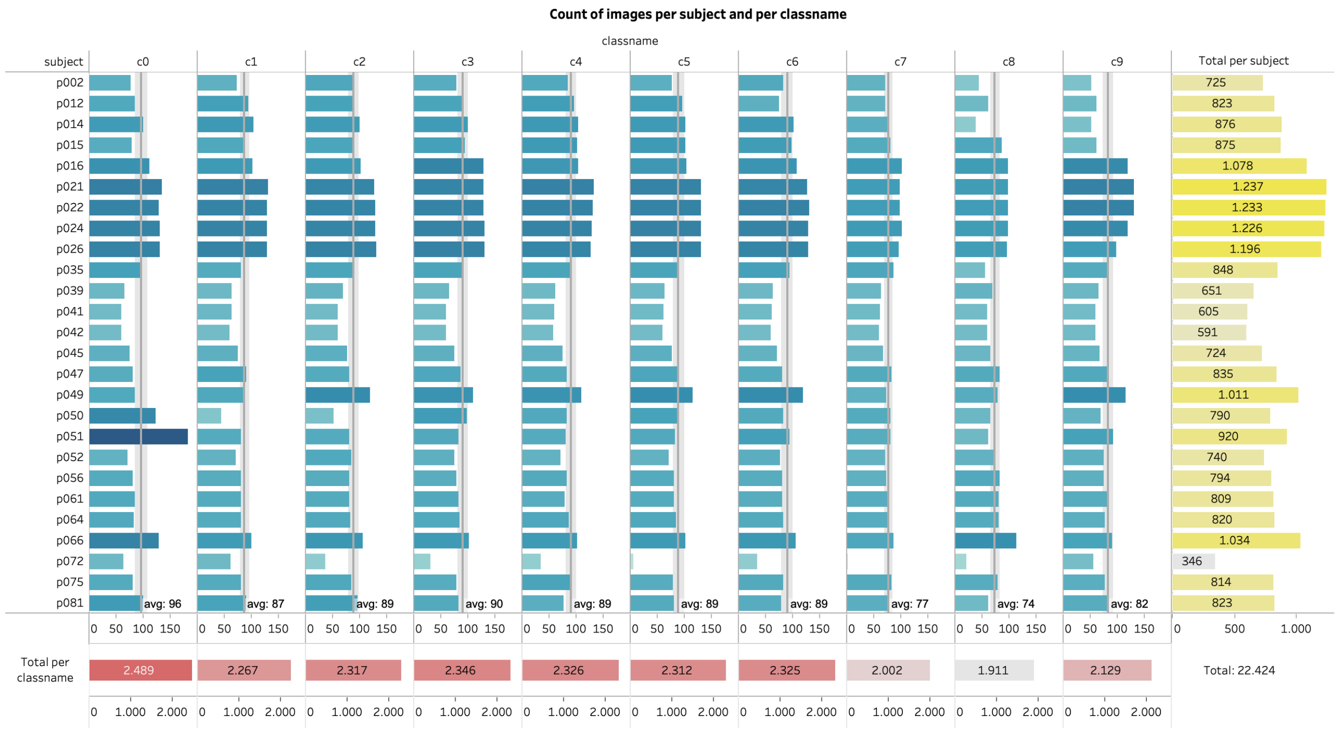

Figure 8: Count of images per subject and per classname present in the Training set.

Appendix B Code & Demo