Engineering Department

Master's Degree Artificial Intelligence and Data Engineering

Università degli studi di Pisa
Accademic year - 2022-2023

Industrial Application

Car driver behaviour detector

STUDENTS:
Tommaso Nocchi, Marco Ralli, Lorenzo Tonelli

Contents

1 Introduction

Many driving accidents are due to driver distraction. The prototype described in the following report considers the development of a model able, through the use of cameras and artificial intelligence, to classify the behavior of a user at the time of driving. The system consists of a set of cameras that will track the user's behavior and a system that will warn of any wrong behavior. Safe-car system includes many security features inside the car, including dog detection, human detection for unconscious abandonment of children or animals inside the car, but also safety features when the machine is moving where the catastrophic consequences can happen very quickly and with little distraction.

Our safe-car system prototype focuses on Pilot recognition, in particular detecting the driver's activities while driving in order to signal dangerous behaviour and avoid problems for people in the car.

2 Dataset

The dataset that has been used to develop and train the networks is retrieved form `https://www.kaggle.com/competitions/state-farm-distracted-driver-detection/data`.

Folders

```
root
├── test
├── train
│   ├── c0
│   ├── c1
│   ├── c2
│   ├── c3
│   ├── c4
│   ├── c5
│   ├── c6
│   ├── c7
│   ├── c8
│   └── c9
├── driverImgList
└── answerSheet
```

From the dir tree is possible to notice the structure of the data. The folder contains a directory with test images but these image are unlabeled so cannot be used for test the network. Train directory contains images that have to be used to train the models, labels are described with *'ci'*. In particular, we have the following classes:

- c0: safe driving
- c1: driver who is texting using the right hand
- c2: driver who is talking on the phone using the right hand
- c3: driver who is texting using the left hand
- c4: driver who is talking on the phone using the left hand
- c5: driver who is operating the radio
- c6: driver who is drinking
- c7: driver who is turning behind
- c8: driver who is wearing makeup
- c9: driver who is talking to passenger

Finally there are 2 files *driverImgList* that has the information about the image that are on train folder and *answetSheet* that is used to insert results of the network once it will be tested with test folder.

This structure is particular because this dataset it had been used for **competition** on kaggle. To use it for our goals two solution has been used. Test set that doesn't present any label is constructed by hand with a sampling of images that are on test folder, roughly 60 image for each class, the other solution used as test set a sampling of train set. All models are tested on the test set sampled from the train set but final score are calculated using ensamble method and during this tests both of them are used.

Figure 1: images examples

In particular, the 10 driving behavior classes found in the dataset have been grouped into 3 macro-classes that will be the labels that our CNN model wants you to classify:
Driving with mobile phone (0), distracted driving (1), correct driving (2)
The class 'c0' of Kaggle will be the correct driving class (2), the classes 'c1' up to 'c4' will be aggregated in 'driving with phone'-class (0) and the remaining 'cX' classes will be collapsed in the 'distracted driving' class (1).
Since our demo had to be tested really by us with photos or videos of correct driving, distracted and with mobile phone and being the data-set found very specific and particular, with images that were very similar to each other, almost the same, we decided to include in the dataset also photos taken by us of the 3 possible behaviors.
In this way we were able to find a balance for the AI model during training, otherwise there would have been the risk of having poor performance due to the big difference among the images that would be passed during the real-time demo and the images on which the model has been trained
An example of photos that were taken manually by the group.

Figure 2: mobile phone driving

Figure 3: distracted driving

Figure 4: correct driving

3 Preprocessing

Image on the train set are not balanced. Each class has different number of samples.

```
CLASS label c0: 2489
CLASS label c1: 2267
CLASS label c2: 2317
CLASS label c3: 2346
CLASS label c4: 2326
CLASS label c5: 2312
CLASS label c6: 2325
CLASS label c7: 2002
CLASS label c8: 1911
CLASS label c9: 2129
```

Figure 5: number of images

Class *'c8'* presents 600 image less than *c0*. So to perform a correct analysis this classes have to be balanced.

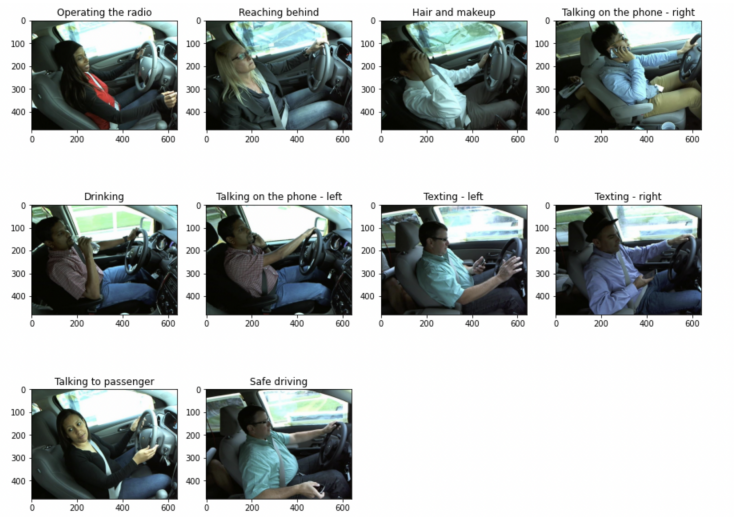


Figure 6: images examples

This are examples of the image that are present on the dataset. These image are very similar between them and the person on the image are often the same. Images that correspond to the same classes are very similar and in some case there are repetitions. Classes *safe driving(c0)* and *talking to passenger(c9)* are difficult to understand also using manual labeling, in fact these two class on the hand test set are often miss classified cause during the creation of the sample probably some mistakes on separate them have been committed.

3.1 Data splitting

First step is to divide the data on train test and validation. This step has been performed as first because on test set cannot be filled with manipulated images, for example images that have undergone geometric or photochromic transformations. As described before two test set are created one using a stratified sampling on the train images and one by hand. After the splitting these are the number of examples on the sets:

```
Num file train c0: 1744
Num file validation c0: 248
Num file test c0: 497
Num file train c1: 1588
Num file validation c1: 226
Num file test c1: 453
Num file train c2: 1623
Num file validation c2: 231
Num file test c2: 463
Num file train c3: 1643
Num file validation c3: 234
Num file test c3: 469
Num file train c4: 1629
Num file validation c4: 232
Num file test c4: 465
Num file train c5: 1619
Num file validation c5: 231
Num file test c5: 462
Num file train c6: 1628
Num file validation c6: 232
Num file test c6: 465
Num file train c7: 1402
Num file validation c7: 200
Num file test c7: 400
Num file train c8: 1338
Num file validation c8: 191
Num file test c8: 382
Num file train c9: 1492
Num file validation c9: 212
Num file test c9: 425
```

Figure 7: images examples

Concerning test set created by hand these are numbers of examples:

```
CLASS label c0: 60
CLASS label c1: 60
CLASS label c2: 60
CLASS label c3: 60
CLASS label c4: 60
CLASS label c5: 60
CLASS label c6: 60
CLASS label c7: 60
CLASS label c8: 60
CLASS label c9: 53
```

Figure 8: images examples

3.2 Data Augmentation and Resizing

The class are still unbalanced so to perform data balancing data augmentation is used to create new synthetic images. To perform data augmentation over the class with less data a random transformations is applied to random images of that classes. A random transformations include different solution, flipping (horizontally and vertically) and rotation (90 180 270). After data augmentation and in case of flipping some image that are by nature **640x440** are transformed in image with a shape **440x640**. To avoid problem during the train of the networks **resizing** is applied to all image in a way to have images that are **300x200**.

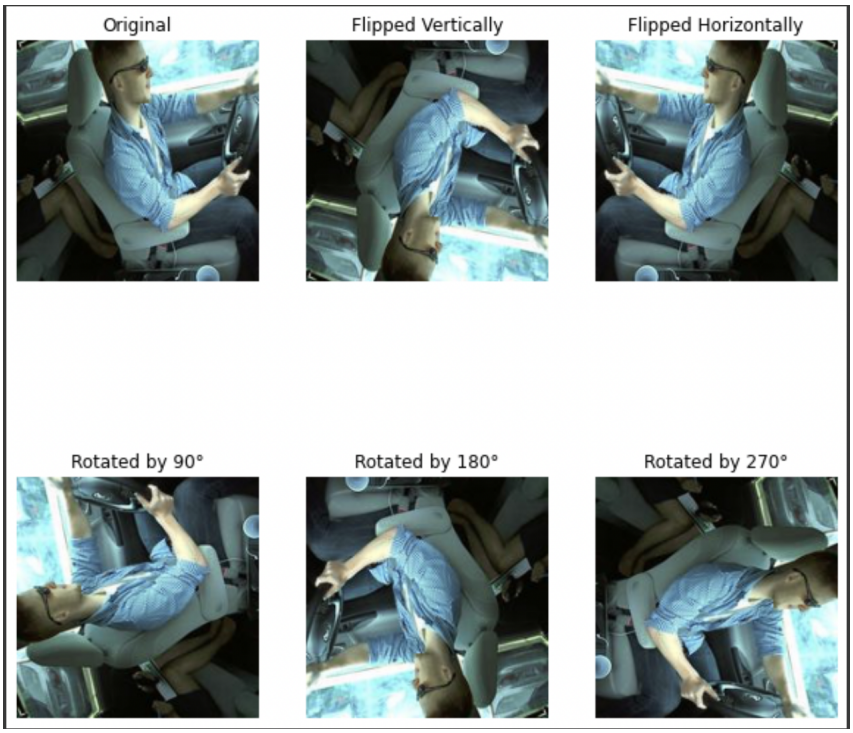


Figure 9: images examples

At the end of the preprocessing the numbers images on the train set are the following:

```
Before augmenting:
1744
After augmenting
1744
Before augmenting:
1588
After augmenting
1744
Before augmenting:
1623
After augmenting
1744
Before augmenting:
1643
After augmenting
1744
Before augmenting:
1629
After augmenting
1744
Before augmenting:
1619
After augmenting
1744
Before augmenting:
1628
After augmenting
1744
Before augmenting:
1402
After augmenting
1744
Before augmenting:
1338
After augmenting
1744
Before augmenting:
1492
After augmenting
1744
```

Figure 10: images examples

4 Pre-trained Models

We decided to use a pre-trained Neural network by exploiting the transfer-learning technique since the CNN are Deep neural network with lots of layers and huge number of trainable parameters that require huge number of samples to be correctly trained from scratch. We decided to use the VGG-16 Neural Network.

4.1 VGG 16

The original architecture of VGG16 is shown in Figure below. The convolutional base consists of five blocks with 2 or 3 convolutional layers whose filters have a size of 3×3. The stack of convolutional blocks is followed by a densely connected classifier consisting of three layers. The final layer is a 1000-node softmax layer. All hidden layers are equipped with the rectification (ReLU). Furthermore, between blocks max-pooling is performed on a 2×2 window, with stride 2. The network expects an input of fixed dimension of 224×224×3 and generates as output a vector of 1000 probabilities corresponding to the 1000 classes in the ImageNet dataset. The input image is assigned to the class with the maximum probability.

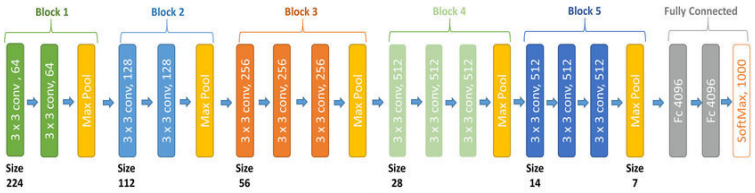


Figure 11: VGG16 architecture

Our experiments on the VGG16 architecture were performed in two stages.

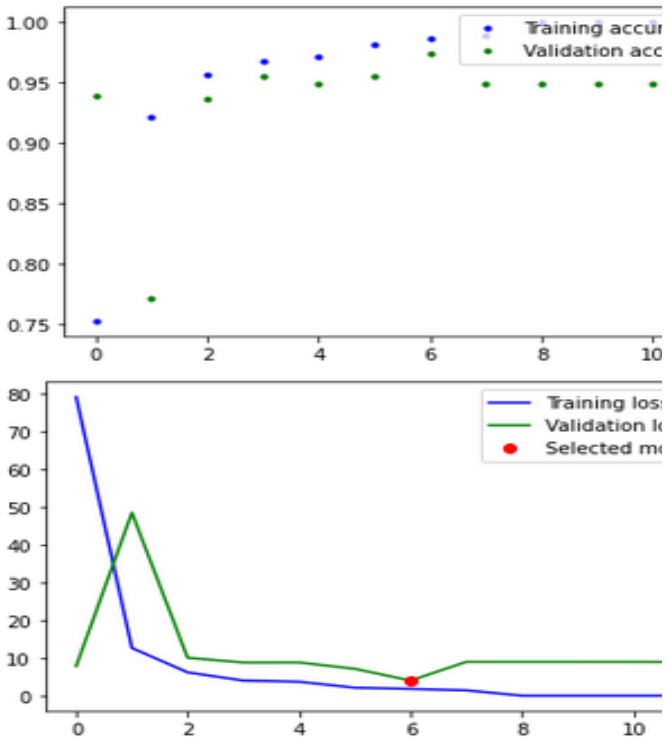
- **Feature Extraction**: we performed a simple feature extraction by taking the convolutional base of the network as it is and training a new classifier on top of its output.
- **Fine tuning**: we fine-tuned the convolutional base, gradually unfreezing the various layers, starting with those closest to the output.

4.1.1 Feature Extraction

The very first thing we tried was to use the convolutional base of the architecture only as a feature extractor. In particular in the experiment we used a dense layer of 256 neurons first without dropout and then with dropout

Without DROPOUT In the configuration without dropout, even if the model succeeds to learn from the training set, it does not perform well on that of validation. In fact, while the training loss is 0.35, the validation loss is 0.8369, which is very high due to the fact that all the conv-base parameters are frozen and trained on imagss of ImagNet which are very different to the ones of our dataset.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.3581	0.9938	0.8369	0.9933

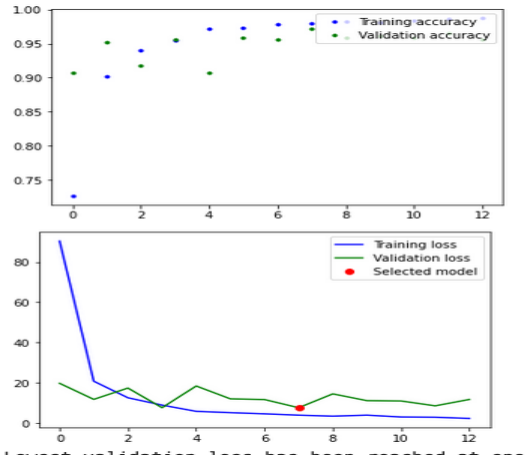



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 180, 180, 3)]     0

tf.__operators__.getitem (S  (None, 180, 180, 3)      0
licingOpLambda)

tf.nn.bias_add (TFOpLambda)  (None, 180, 180, 3)      0

vgg16 (Functional)           (None, 5, 5, 512)         14714688

flatten (Flatten)            (None, 12800)             0

dense (Dense)                (None, 256)               3277056

dense_1 (Dense)              (None, 10)                2570

=================================================================
Total params: 17,994,314
Trainable params: 3,279,626
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 12: Learning curves and summary of the network

With DROPOUT In the configuration with dropout the result achieved are worse than the configuration without dropout, this due to the fact that we were using less neuron in the upper layers with respect to the previous experiments in order to reduce the co-adaptation and to encourage the neurons to find, in the optimal way, the true nature of the input-output mapping.

Training loss	Training accuracy	Validation loss	Validation accuracy
2.7864	0.9705	1.0860	0.9920



Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 180, 180, 3)]	0
tf.__operators__.getitem_1 (SlicingOpLambda)	(None, 180, 180, 3)	0
tf.nn.bias_add_1 (TFOpLambda)	(None, 180, 180, 3)	0
vgg16 (Functional)	(None, 5, 5, 512)	14714688
flatten_1 (Flatten)	(None, 12800)	0
dense_2 (Dense)	(None, 256)	3277056
dropout (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2570

Total params: 17,994,314
Trainable params: 3,279,626
Non-trainable params: 14,714,688

Figure 13: Learning curves and summary of the network

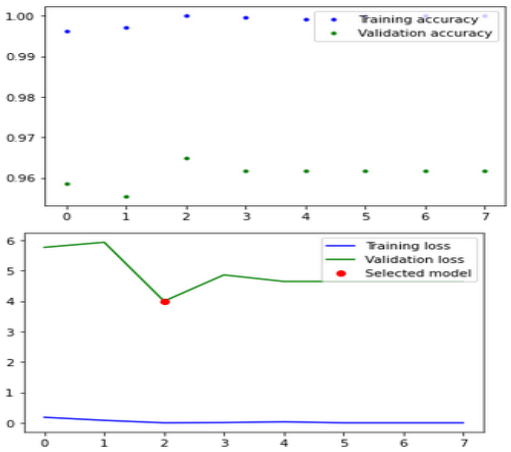
In both the experiment we noticed good value with the test set, as we can see from the confusion matrix, most of the test sample have been correctly classified.

4.1.2 Fine Tuning

Since the highest layers of the network have most likely learned to recognize features that are more specific to the original dataset, we though that performing fine-tuning might be beneficial to the performance of the model, but apparently the model start to suffer of overfitting as we can notice from the loss plot. The overfitting is caused by the big difference in the number of trainable parameters that are passed from 3,279,626 to more than 10 million.
In this experiment we have taken the model from the configuration without dropout since it has given better result than the one with dropout and we unfrozen, first, only the last convolutional layer of the convolutional base and we trained the resulting model with our training set with a very low learning rate.

Unfreeze only the last convolutional layer of the VGG16 The accuracy values reflects good results but there is a big difference in the loss plot among training and validation, the main reason is the huge increment of trainable parameters in comparison with the size of the dataset. The confusion matrix shows good results in the prediction of the test set being images very similar to the ones presents in the training set

Training loss	Training accuracy	Validation loss	Validation accuracy
0.0180	0.9994	0.7896	0.9924

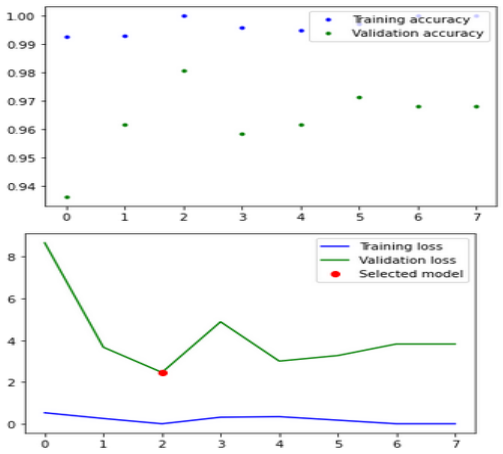



```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 300, 200, 3)]     0

tf.__operators__.getitem_3   (None, 300, 200, 3)       0
(SlicingOpLambda)

tf.nn.bias_add_3 (TFOpLambd  (None, 300, 200, 3)       0
a)

vgg16 (Functional)           (None, 9, 6, 512)         14714688

flatten_3 (Flatten)          (None, 27648)             0

dense_8 (Dense)              (None, 128)               3539072

dense_9 (Dense)              (None, 3)                 387

=================================================================
Total params: 18,254,147
Trainable params: 10,618,883
Non-trainable params: 7,635,264
_________________________________________________________________
```

Figure 14: Learning curves and summary of the network

Unfreeze last two convolutional layers of the VGG16 Subsequentally starting from the model without dropout we unfrozen the last two layers convolutional layers of the convbase. The resulting model starts to suffers of overfitting, as we can see from the plot, due to the very big amount of trainable parameters, very close to 17 million.
However also this fine-tuning configuration improve the performances w.r.t the starting model by ahcieving the lowest validation loss value of the experiments with VGG16.
Also in this configuration the model obtained from VGG16 has provided good results with the test set. The main reason is the size of the dataset set and the quality of the images composing it.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.1010	0.9978	0.7109	0.9911




```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 300, 200, 3)]     0

tf.__operators__.getitem_3   (None, 300, 200, 3)       0
(SlicingOpLambda)

tf.nn.bias_add_3 (TFOpLambd  (None, 300, 200, 3)       0
a)

vgg16 (Functional)           (None, 9, 6, 512)         14714688

flatten_3 (Flatten)          (None, 27648)             0

dense_8 (Dense)              (None, 128)               3539072

dense_9 (Dense)              (None, 3)                 387

=================================================================
Total params: 18,254,147
Trainable params: 16,518,659
Non-trainable params: 1,735,488
_________________________________________________________________
```

Figure 15: Learning curves and summary of the network

4.2 Final Test

We decided to perform the final comparison by using the confusion matrix obtained form the test set in order to select the model that has been able to generalize better than others.

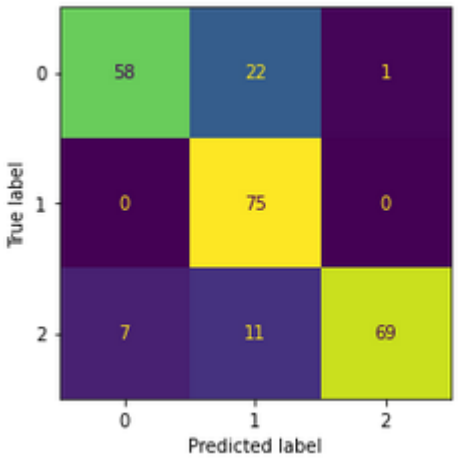


Figure 16: Fine-Tuning 1

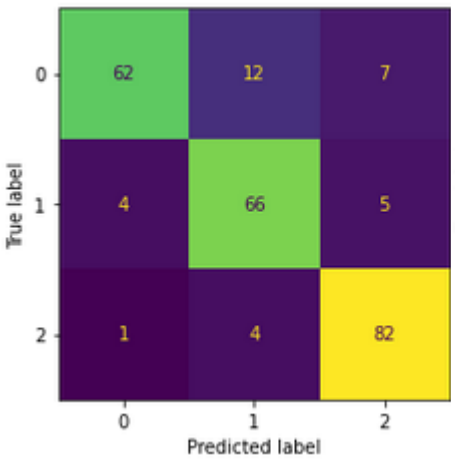


Figure 17: Fine-Tuning 2

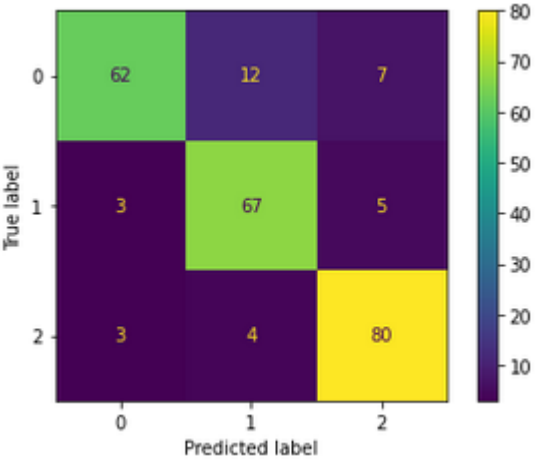


Figure 18: VGG without regularization and fine-tuning

We decided to choose as final model the second fine tuning configuration since it has provided the best confusion matrix values. As we can see from the confusion matrix the false positve and false negative are very low for all the tested models but the second configuration of fine tuning has provided the best values

5 Explainability

We tried to understand how out networks classify the images. There are a lot of possible techniques, we decided to focus on Intermediate Activations and Heatmaps of the class activations.

5.1 Intermediate activations

Visualizing intermediate activations consists in displaying the feature maps that are outputted by the convolutional and pooling layers in the network, given a specific input. To see the differences between the models we will consider the same image, which is the following.

Figure 19: Test image

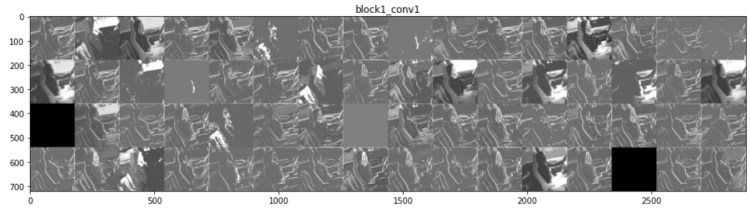


Figure 20: Intermediate activations VGG16

In this case there are both filters dedicated only to edge analysis and filters that focus on the background outside of the cell. This appears to be an intermediate behavior to that seen for the other networks. Of course, there are also filters that consider internal details already in the first convolutional layer.

5.2 Heatmap of class activation

This technique is useful to understand which parts of a given image led a convnet to its final classification decision. We will briefly analyze the behavior of the networks when they receive as input an image of a safe drive (class C0) and an image of a driver who is texting with his right hand (class C1). So we focused in

compare a safe and an unsafe driving, using a CNN from scratch network and a pretrained on inception model network.

5.2.1 Safe driving

Both the heatmaps were computed based on the same image to get comparable results.

Figure 21: Safe driving input image

From the input image we got the corresponding heatmaps for the considered models.

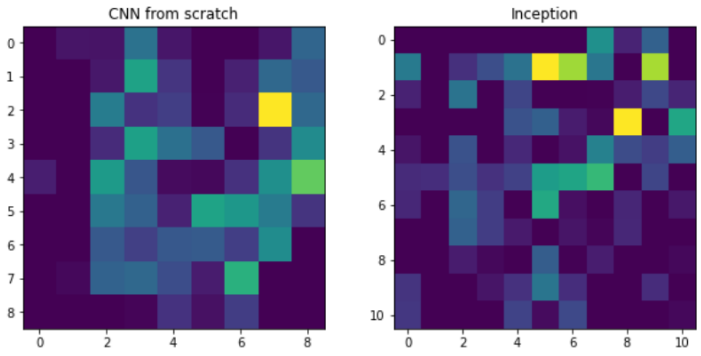


Figure 22: Input image heatmap

Then we superimposed the heatmaps to the input image, obtaining the two following pictures.

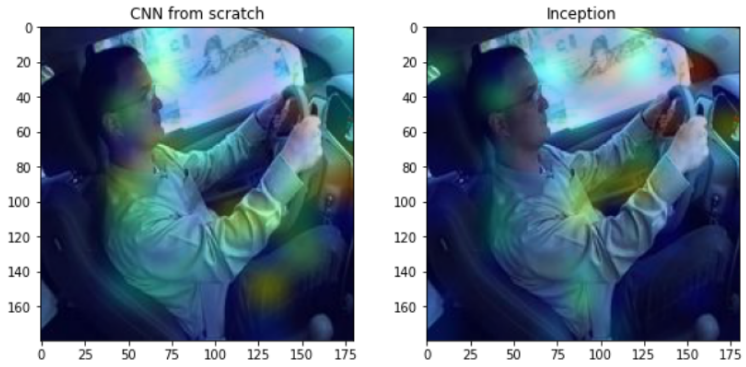


Figure 23: Input image heatmap

From the heatmaps, in both cases we can see that the models are able to classify the input image as belonging to class of safe driving focusing on the hands positions, indeed the driver keeps his hand on the wheel, and on the hand position which is directed towars the road.

5.2.2 Unsafe driving

Both the heatmaps were computed based on the same image to get comparable results.

Figure 24: Unsafe driving input image

From the input image we got the corresponding heatmaps for the considered models.

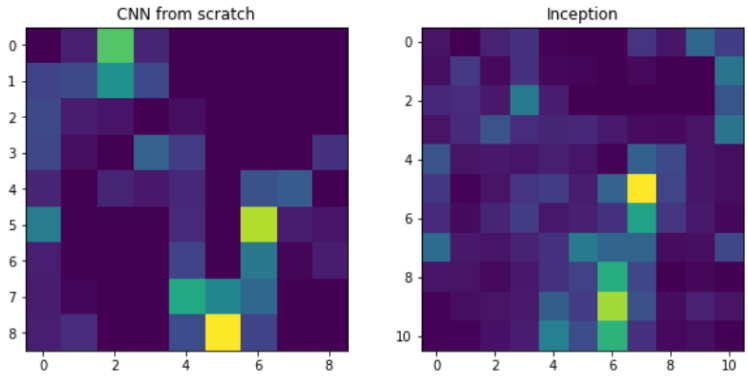


Figure 25: Input image heatmap

Then we superimposed the heatmaps to the input image, obtaining the two following pictures.

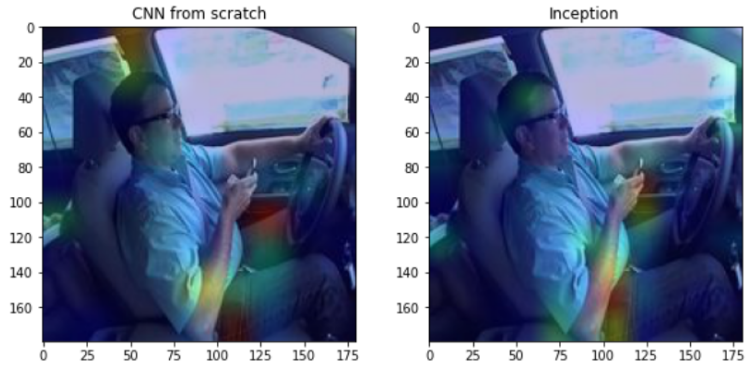


Figure 26: Input image heatmap

From the heatmaps, in both cases we can see that the models are able to classify the input image as belonging to class of safe driving focusing on the right hand which is holding the phone and on the right elbow.

6 Demo Implementation

The prototype consists of a Raspberry PI board to which you can mount a camera and via TCP connection with the PC will transmit images to be classified by a Deep Learning model, more specifically a Convolutional Neural Network, ideal for image application.
In particular we want to classify when the driver is driving correctly (0) or incorrectly, such as use of the phone with left hand or right hand during the drive (1), the use of radio (2), or while the driver is drinking or talking with passengers (3).

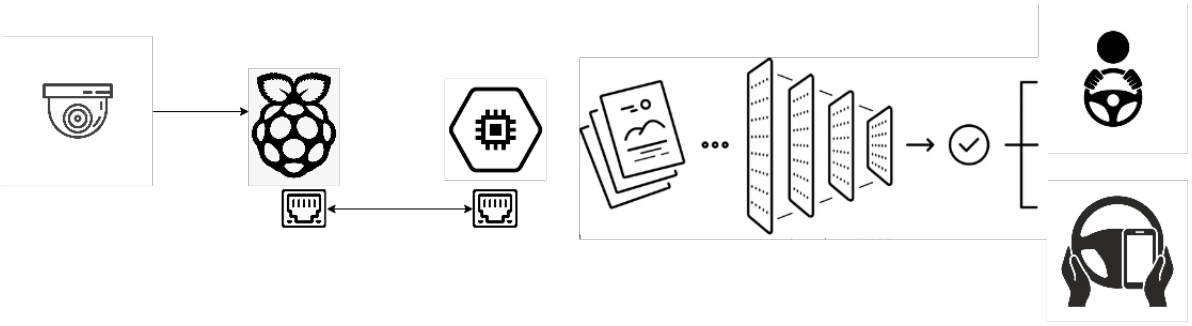


Figure 27: prototype schema

The photos of the driver are sampled from short video that last 5 or 10 seconds through the use of a camera connected with a Raspberry PI board which in turn is connected via Ethernet to PC.
The camera is placed on the opposite side of the driver in order to capture with more accuracy every action that the driver can perform.

6.1 Description of the system

The hardware used within the system are:

- **PC Dell Inspiron 13 700**
  16GB RAM, Intel UHD Graphic, Windows 10, Intel Core i7-10510U CPU
- **Anaconda platform**
- **Power supply of 2.4 A, Cavo ethernet rj45, Raspberry Camera**
- **Rapsberry PI 3 model b+**
  Quad Core CPU, 1GB RAM, 4 porte USB 2, 128GB microSD (added separately)

6.2 Libraries

To link all the component of the system the following libraries have been used:
Raspberry:

- **socket**
  to implement the TCP communication client server
- **OpenCV**
  multiplatform software library for real-time machine vision

- **OS**
  OS module of the Python language has several useful functions to make the program interact with the operating system of the computer (Windows, Linux or Mac OS).
- **Picamera**
  This package provides a pure Python interface to the Raspberry Pi camera module for Python 2.7 (or above) or Python 3.2 (or above).

Application

- **numpy**
  Python library used for working with arrays(images representation)
- **tensorflow**
  free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
- **OS**
  OS module of the Python language has several useful functions to make the program interact with the operating system of the computer (Windows, Linux or Mac OS).
- **PIL**
  The Python Imaging Library adds image processing capabilities to your Python interpreter.
- **palysound**
  pure Python, cross platform, single function module with no dependencies for playing sounds

6.3 Algorithm for classification

Images that arrived from the raspberry cam are collected within a folder "image". Once 10 images has been collected the algorithm start to classify them.

Data: $imageFolderPath, classificationModel$
$flagSafe, flagPhone, flagDistraction = 0$;
for $image \in imageFolderPath$ **do**
  $result \leftarrow getClassification(classificationModel, image)$;
  if $res ==$ "$phone$" *is true* **then**
    $flagPhone++$;
  else if $res ==$ "$distraction$" *is true* **then**
    $flagDistraction++$;
  else
    $flagSafe++$;
  end
end
return max(flagSafe, flagPhone, flagDistraction)

Algorithm 1: Classification of image

6.4 Code Description

Our Demo works according to the client/server paradigm. The client part is performed by the raspberry which plays the role of data acquisition, while the more substantial part of the business logic, image classification, voting algorithm and audio reproduction is entrusted to the server which resides in the PC. In this section of the documentation we want to give a description to the code, describing separately the code for the client and the one for the server. The code can be consulted directly on the github repository by clicking on the following link: Demo Repository

6.4.1 client side

The raspberry plays the role of acquiring data (images) which are periodically sent to the server. The IP address and port are required to connect to the server. To find out the IP address of the eathernet device on the PC side, the Windows command $ipconfig all was used. A second necessary parameter for the execution of the client code was the resolution of the camera in capturing the photos. It was not necessary to use high resolutions since our neural network works with low image resolutions. A third parameter to choose fell on the frequency of time with which to take the photos. After several experiments it was decided to send one photo per second to the server. This is due to the fact that the neural network is slow in classifying images, so it would have been useless to send too many photos to the server side.

```
1
2  HOST = "169.254.108.159"
3  PORT = 65432  # The port used by the server
4
5  camera = PiCamera()
6  camera.start_preview()
7  camera.resolution = (640, 480)
8  delta_time = 1.0
9  sleep(10)
10 image_path = ''
11 i = -1
12 while(True):
13     i = i + 1
14     sleep(delta_time)
15     image_path = '/home/pi/Desktop/image/image%s.jpg' %i
16     camera.capture(image_path)
17
18     # Load image (it is loaded as BGR by default)
19     image = cv2.imread(image_path)
20     # Conver array to RGB
21     #image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
22     # image encoding
23     success, encoded_image = cv2.imencode('.jpg', image)
24     # convert encoded image to bytearray
25     content_bytes = encoded_image.tobytes()
26     print("dimensione immagine: ",len(content_bytes))
27
```

```
28      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
29          s.connect((HOST, PORT))
30          s.send(content_bytes)
31
32      print("photo sent")
33
34  camera.stop_preview()
```

6.5 server side

The business logic is executed completely in the code that runs on the server. We did not want to entrust the running of the neural network to the raspberry, which also did not have sound cards to play sounds.

The first part is based on defining the variables necessary for the functioning of the code. flag_safe, flag_dist and flag_phone are counters that count how many of the last n photos have been classified by the network as safe, distraction and phone respectively. An important parameter is the NPR, Number of Photo Received sequentially. Indicate how many photos are needed to determine the driver's driving status. This is a very important value: too low a value would result in being subject to false positives; too high a value would require not capturing a driving state of the driver but more driving states of the driver returning an incorrect classification of the network.

```
1
2  if __name__ == "__main__":
3
4      count = 0
5      NPR = 0
6      flag_safe = 0
7      flag_dist = 0
8      flag_phone = 0
9      print("System is going to start ------>\n")
10     print(".......\n")
11     dest_folder = 'image/'
12
13     vgg_model = tf.keras.models.load_model('model/model.h5')
```

In the instruction of line 3 we wait for the receipt of the image, which, once received, is saved in a specific folder. This photo is classified and the appropriate counter is increased based on the result of the network. In line 10 we check if enough photos have been received to determine the driving status of the driver.

```
1   while(True):
2
3           path_img = server_tcp.get_raspberry_image(dest_folder)
4           img = cv2.imread(path_img)
5           cv2.imshow('output',img)
6
```

```
7           if cv2.waitKey(20) & 0xFF == ord('q'):
8               break
9
10          if len(os.listdir("image")) == NPR:
11              for image in os.listdir("image"):
12                  result = classification.classification(vgg_model,"image/"+image)
13                  print("Result: " + result)
14                  if  result == "Phone":
15                     flag_phone = flag_phone + 1
16                  elif result == "Distraction":
17                      flag_dist = flag_dist + 1
18                  else:
19                      flag_safe = flag_safe + 1
20                  os.remove("image/"+image)
21                  num_image = 0
```

In the last part we are going to play the appropriate audio based on the major type of driving state that has been identified in the latest NPR images received. After playing such audio, the counters are reset to 0.

```
1
2               if max(flag_phone, flag_dist, flag_safe) == flag_dist:
3                   playsound('audio/dist.mp3')
4               elif max(flag_phone, flag_dist, flag_safe) == flag_phone:
5                   playsound('audio/phone.mp3')
6               elif max(flag_phone, flag_dist, flag_safe) == flag_safe:
7                   playsound('audio/safe.mp3')
8
9               flag_safe = 0
10              flag_phone = 0
11              flag_dist = 0
```

7 Future improvementes

Our demo is still far from being marketable. Drive monitoring system is a category where high performance is required and the system falls under soft real-time systems.

In a real context it is unthinkable to make the network work on a computer without a GPU. The performance and some specifications can be met if we have a pc with a good integrated GPU.

Computing power is not enough to increase network performance. Integrating our network with one that is able to reveal human emotions through micro facial expressions can have a huge impact by decreasing the number of false positives or false negatives provided by the network.