ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Machine Learning for Computer Vision

AN EMBEDDED IN-CABIN LIGHTWEIGHT HIGH-PERFORMANCE 3D GAZE ESTIMATION SYSTEM

CANDIDATE
Angely Oyola Suárez

SUPERVISOR
Francesco Conti, PhD

CO-SUPERVISOR
Prof. Samuele Salti

Academic year 2022/23

Session 1st

To my love, my mother, who introduced me to computers
without knowing that I was going even crazier.

Contents

1 Introduction and Related Work **1**

1.1 Summary 1

1.2 Gaze Estimation On the Edge 3

1.3 Thesis Goals 5

1.4 Thesis Contributions 5

1.5 Thesis Structure 6

2 Dataset Collection and System Overview **7**

2.1 Dataset 8

2.2 System Overview 11

2.2.1 Face Detector by means of Facial Landmarks 11

2.2.2 Drowsiness detection 17

2.2.3 3D Gaze Estimator 18

2.2.4 Deployment on the edge 31

2.2.5 Gaze Estimation in live 40

2.2.6 Gaze Zones Calibration 45

3 Experimental Results **46**

3.1 Evaluation 46

3.2 Live Demo 49

3.3 Experiments 51

4 Discussion and Conclusions **56**

4.1 Final Remarks 56
4.2 Limitations 57
4.3 Future work 58

Bibliography **60**

Acknowledgements **71**

List of Figures

1.1 Appearance-based deep learning neural network to estimate 2D and 3D gaze. Workflow of [86] 3

2.1 Acquisition set-up adopted by [38] 8
2.2 Subject's and camera position adopted by [38] 10
2.3 Flowchart of the proposed system 12
2.4 Eye landmarks returned by MediaPipe Face Mesh [27] 19
2.5 Gaze Estimation in difficult scenarios. Whereas a) and b) appear to be the same image, their gaze vectors are different. Then, the head pose helps to distinguish where she is looking. 21
2.6 ReLU6 activation function of MobileNetV2 [22] 23
2.7 Depth-wise separable convolution introduced by MobileNets [33] 24
2.8 a) Traditional residual block and b) Inverted Residual Block proposed on MobileNetV2. Figure from [66] 25
2.9 Architecture of MobileNetV2 as backbone of the proposed model [66] 26
2.10 Pinball Loss 29
2.12 Split of the cabin into Gaze Zones 31
2.13 Split of the cabin into quadrants. It is based on Yaw (θ) and Pitch (ϕ) angles 32
2.14 Deployment on the edge pipeline adopted by Pytorch Mobile [61] 33

2.15 IEEE 32-bit floating point representation [70] 34
2.16 Quantization Aware Training (QAT) Pipeline 38
2.17 Pytorch transition from Eager to Script 38
2.18 Raspberry Pi4 model B used in the experiments of this work . 41
2.19 Raspberry Pi4 case 42
2.20 Materials used on the live demo 42
2.21 Camera set-up adopted by this work 44
2.22 Outside view of the embedded system created in this work . . 44

3.1 Comparison of the Proposed Model with the top-3 models reported on the Gaze360 dataset benchmark. In the x-axis the number of operations [GigaOPs]. In the y-axis the Angular Error [degree]. The size of each figure varies according to the number of parameters. 48
3.2 Live demonstration of the system proposed in this work. A 3D gaze estimator deployed on a Raspberry Pi4. The system is able to detect: the driver's face and head orientation using 3D facial landmarks, the 3D gaze and nine in-cabin gaze zones. 50
3.3 Flowchart followed to quantize a model using Tensorflow Lite 55

List of Tables

2.1 Performance of the Facial Landmark Detector on two mobile devices[86] . . . 14
2.2 MobileNetV2 Classification Head . . . 25
2.3 Pinball Loss Quantiles . . . 28
2.4 Gaze zone description . . . 31
2.5 List of materials used to perform gaze estimation on Raspberry Pi4 . . . 40

3.1 Comparison of performance of the model in their FP32 and optimized versions. **1st row:** Eager FP32 model, **2nd row:** INT8 (QAT + Layers Fusion) and the **last row:** the proposed model Pytorch JIT Script . . . 47
3.2 Performance of the different models created with ResNet18 as feature extractor and comparison with the FP32 version of the proposed model that makes use of MobileNetV2 as feature extractor . . . 53

Chapter 1

Introduction and Related Work

1.1 Summary

Human gaze has been revealed as an important cue to obtain information about internal cognitive state [82], such as intention detection and visual attention. Therefore, human gaze estimation, also known as eye tracking, has become an active field of computer vision research in recent years. It has been applied to different fields, some of them: human robot interaction [77] [26], augmented reality [35], medicine [30], aviation [58], and automotive [53] [81].

In the latter field, gaze estimation has been a key part of Advanced Driver Assistance Systems (ADAS), enabling the development of cutting-edge systems that ensure safe driving, mitigate road accidents, facilitate autonomous driving and provide comfort. This thesis will focus on the development of a lightweight and low-cost in-cabin gaze estimator capable of operating at the edge. Although the main objective is safety related by monitoring driver attention, the system can be extended to other purposes as well.

Gaze estimation can refer to the detection of the 2D device's coordinates (x,y) intersecting the gaze vector, known as the point of gaze (PoG), as well as the estimation of the 3D gaze direction in the camera reference system [84], as shown in 1.1. To recover the 3D gaze vector, this thesis will focus on the latter group .

When estimating the 3D gaze vector or 2D gaze point, there are different approaches that can be followed. They can be classified as model-based or appearance-based.

To detect gaze, the first group analyzes the geometry of the eye and some of its parts, such as: pupil center, pupil outline, iris, corneal reflection, and glint [16]. Although they can obtain high accuracy, they are limited to dedicated devices such as infrared and depth sensors, which may incur high investment and low user experience due to physical intrusion [36].

In contrast, the appearance-based group does not analyze eye geometry, but only appearance. They are based on gaze direction regression of features extracted from images containing eyes or a face. Without the need for dedicated devices, appearance-based estimators are also known as unconstrained gaze estimators due to their calibration-free, subject-free and viewpoint-independent [15] architecture. With respect to performance, they can obtain good accuracy with only one RGB camera, a needs-covering human gaze dataset, a well-performing feature extractor, and an effective regression function to map the appearance of the eye - the human gaze [16] [15] (see Figure 1.1).

Despite the importance of this topic, only a small number of works in computer vision are concerned with gaze estimation [64]. Although most of them are capable of running in real time and with high performance, they are limited to the use of dedicated hardware and, in most cases, expensive devices such as infrared sensors, depth cameras, and GPUs. In the automotive field, the driver's head and eyes are key components for inferring gaze and visual attention at the same time [78]. Several use cases of gaze estimation have been reported in this field [44] [4] [81]. They focus on solving this task to improve safety and/or comfort by: drowsiness detection, driver distraction [1], emotion recognition [46], degenerative disease detection [10], ADAS enhancement [39], autonomous driving [8], and augmented reality-based systems [45]. Although high-performance solutions exist, only a few are non-intrusive embedded systems based on deep learning.

In this work, a low-cost, lightweight and high-performance embedded system based on deep learning, capable of estimating the 3D gaze vector and the in-cabin gaze zone where the driver is looking, while detecting drowsiness, is proposed. The system falls into the group of appearance-based 3D gaze estimation by analyzing only RGB images using deep neural networks. Although gaze-zone detection can be treated as a classification problem rather than regression, our proposal is based on the latter group for the following reasons 1) Lack of large datasets for in-cabin gaze zone detection. 2) The calibration and training required each time a new gaze zone is added. 3) The advantages of having the gaze vector in 3D.

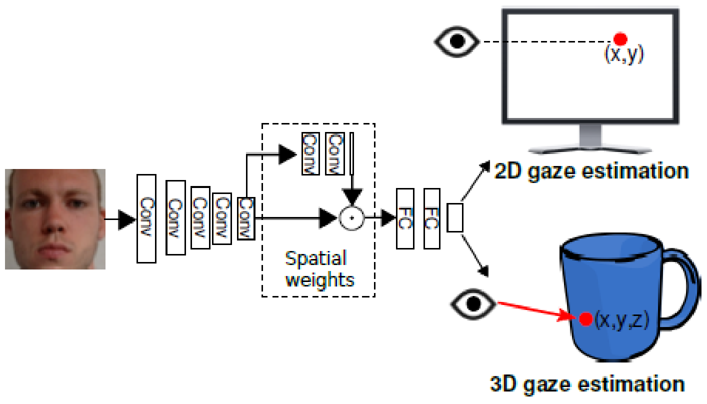


Figure 1.1: Appearance-based deep learning neural network to estimate 2D and 3D gaze. Workflow of [86]

1.2 Gaze Estimation On the Edge

With the current rise of IoT technologies, the deployment and direct inference on the edge (MCUs and embedded systems in general) of machine learning models, including those based on deep learning, has gained interest [75].

Although there are several gaze estimation systems that offer high performance and real-time inference, most of them are very limited to work with dedicated and, in most cases, expensive devices, such as: infrared sensors, depth cameras and GPUs. So far, there are few works estimating 3D gaze on edge devices using an RGB camera as the only sensor.

Improving the performance of deep learning models running on GPUs involves changes to the model architecture that, in most cases, increase its size and computational complexity. However, when deployed at the edge, this is something that is not acceptable due to the limited computational power and memory of the devices used [32].

Therefore, running models on the edge requires, in most cases, a trade-off between model size, complexity, and memory footprint with performance and inference time [11]. There exist efficient architectures such as MobileNets [33], ShuffleNet [83] and MobileVit [49] that allow us to inference on mobile devices in real time and with high performance when solving general tasks such as Image Classification, Segmentation, or Object Detection.

However, when developing specific task-oriented models, even more if it refers to two deep-learning-based models working in sequence, as is the case of this work, achieving a high performance with those architectures will be, in most of cases, only possible at expense of the inference time.

For that reason, before deploying on device, the model needs to pass through an optimization process using a set of methodologies such as quantization, fusion of layers, distillation, pruning, and clustering that allow to perform computation at lower bit-widths than floating point precision, and to speed up the inference time by dropping some model parameters for keeping a low impact in the performance.

Machine learning on the edge, also known as TinyML has the main goal of developing efficient high-performance systems without the need of specialized hardware. It matches the main goal of this work, to come up with a low-energy, low-cost and high-performance on-device system, that is able to detect in short time where the driver is looking at through 3D gaze estimation.

1.3 Thesis Goals

The present thesis aims to achieve the following goals:

- develop a low-cost, efficient and high-performance on-device system capable of estimating the driver's gaze using deep learning and an RGB camera.
- identify the in-cabin gaze zone where the driver is looking.
- detect possible drowsiness.

1.4 Thesis Contributions

In this work, an end-to-end embedded system based on deep-learning is proposed to estimate in-cabin 3D gaze. The main contributions are as follows:

- creation of a low-cost embedded prototype of an efficient appearance-based 3D gaze estimator capable of detecting possible drowsiness, driver's 3D gaze and the in-cabin gaze zone using a Raspberry Pi 4 and an RGB camera.
- creation of a gaze estimation workflow by assembling two deep-learning-based models. The first one detects the driver's face using 3D facial landmarks, while the second one estimates the 3D gaze vector. Finally, using the gaze vector, the in-cabin gaze zone is classified.
- application of optimization techniques, such as quantization and layers fusion, to reduce model size and speed up inference while maintaining good performance.

- experiments trying different deep-learning-based models on the role of the backbone and regression head, optimization techniques and recommendations of related work. This analysis may be useful for researchers interested in this field.

1.5 Thesis Structure

This thesis is organized as follows: Chapter 2 covers the dataset and overview of the proposed system. Chapter 3 describes the evaluation of the gaze estimation model, the live demonstration and the experiments performed. Conclusions, limitations of the proposed system and future development are found in Chapter 4.

Chapter 2

Dataset Collection and System Overview

Though there are several open-source datasets for training deep-learning-based gaze estimators such as: MPIIFaceGaze [86], EYEDIAP [24] and Columbia [71], only few of them cover different variation in illumination, subject and distance from the camera, as well as, wider head pose, yaw (θ) and pitch (ϕ) gaze ranges.

Even if the driver's gaze will be hardly ever 360[°] on both θ and ϕ, however, it has been used the largest subject, pose and illumination variant open-source dataset for unconstrained 360[°] 3D gaze estimation, Gaze 360 dataset [38], to train the proposed model.

More details of the dataset creation, including the camera set-up and the equations used to calculate the 3D gaze vector, which will be used as ground truth to train and validate the proposed model, will be found starting from 2.1.

2.1 Dataset

Set-Up

Gaze 360 dataset has been created using a LadyBug 5 360[°] panoramic camera. The camera is composed by 5 units of 5 [Mpx] with a horizontal range view of 120[°] each one and another upward-facing unit with the same characteristics but not used for gathering data. The portable set-up is shown in 2.1 and a full face can be detected by at least one of the units if the person is standing far away from the camera at least 1 meter.

All frames have been rectified to remove barrel distortion. Later, the head and feet keypoints have been detected on each rectified frame using AlphaPose [23]. All participants have been indicated to look at the only target present in the set-up, an AprilTag [76] mobile board with a cross on it. AprilTag is a visual tag detector widely used for its robustness and efficiency in fields such as robotics and computer vision. The visual tag has been used to make possible the 3D gaze tracking, while the cross to get gaze fixation.

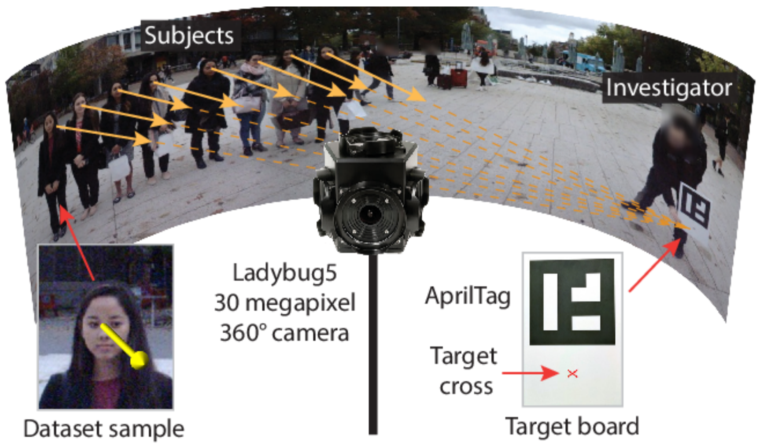


Figure 2.1: Acquisition set-up adopted by [38]
to create Gaze360 dataset

3D Gaze Estimation

In order to estimate the 3D world reference coordinates of the target cross p_t, it has been firstly estimated the 3D pose of the visual tag using both the already known camera calibration parameters and the tag size. Already known the tag 3D coordinates and the board geometry, the 3D coordinates of the p_t have been calculated.

The gaze vector is expressed in the camera coordinates systems L and is calculated as $g_L = p_t - p_e$. In order to estimate the position of the subject's eyes p_e, it has been calculated the distance *d* of the subject's eyes from the camera by means of trigonometry 2.2. The step by step followed is detailed below:

- it has been assumed that both the subjects and the camera set-up lay on the same horizontal plane.
- α and β are the spherical coordinates of eyes and feet respectively and have been estimated using both the keypoints detected by AlphaPose and the Global Cartesian 3D camera system $L = [L_x, L_y, L_z]$.
- with the already known camera height *h*, α and β, it has been calculated *z* using 2.1. It represents the subject's feet distance from the camera.
- finally, *d*, the distance of the subject's eyes distance from the camera has been calculated as 2.2.

$$\mathrm{z} = h.\tan(\frac{\pi}{2} - \alpha) \tag{2.1}$$

$$\mathrm{d} = \frac{z}{\cos\beta}) \tag{2.2}$$

If the gaze vector is calculated in the camera coordinates system L as mentioned before $g_L = p_t - p_e$, it would suffer of changes with the rotation of the camera and the variation of L. Therefore, the gaze vector has been expressed

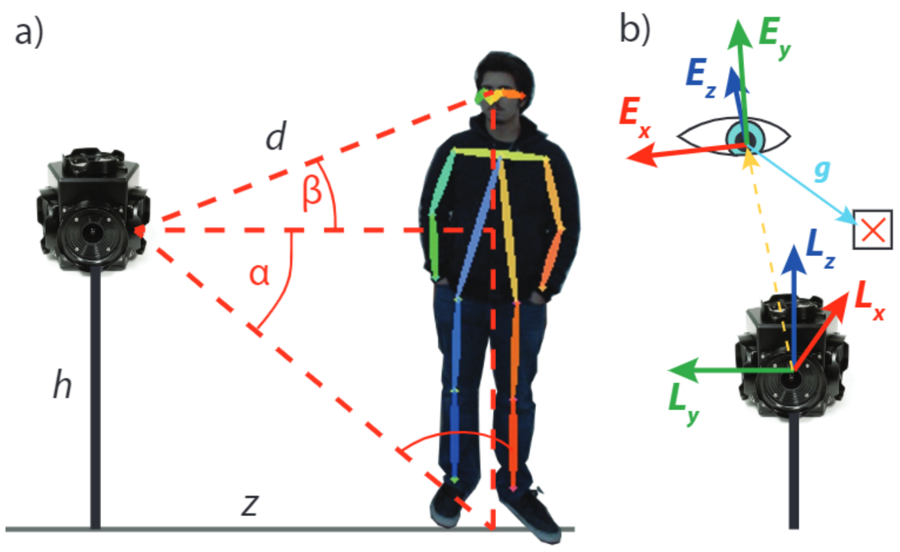


Figure 2.2: Subject's and camera position adopted by [38] to estimate a) the subject's eyes distance from camera and the gaze transformation b) from the camera system to the subject's eyes system.

in the subject's eyes coordinate system E. As the origin of E it has been defined p_e.

In fig. 2.2 can be appreciated that E_z has the same direction as g_L, L_x and L_y without roll define the plane where E_x lies on. Thus, the gaze vector in the eyes' coordinates is defined as:

$$g = E.(\frac{g_L}{||g_L||_2}) \tag{2.3}$$

Following the equation 2.3, when the subject looks directly the camera, the gaze will be $g = [0, 0, -1]$.

It is worth to mention that a control experiment was applied in order to validate the accuracy of the calculated gaze, those that have be used as ground truth to train the model proposed on this work. The experiment has carried with a single participant and two as total with three recordings by each one. Together with the Ladybug 5 360 [°] camera, it has been used an additional front-facing test camera placed over the participant's right eye. Later, it was estimated the gaze on both cameras, applying the equations mentioned before. The validation resulted in a mean difference between both gaze labels of 2.9[°].

2.2 System Overview

The flowchart of the proposed system is shown in Figure 2.3. The proposed system is composed by two deep neural networks models working sequentially. The first corresponds to a ready-to-use face detector model (API) that receives the driver's frontal RGB image as input, that in normal circumstances, it will include the driver's face. Then, based on a proposed 3D facial landmarks analysis, the face will be identified and cropped from the original image. The second model corresponds to the gaze estimator. It receives the already cropped face as input and estimates the 3D gaze vector. Finally, the 3D coordinates of the gaze are converted into spherical coordinates and classified into a gaze zone.

Regarding the programming language and deep learning framework, it has been used Python 3.7.13 and Pytorch 1.11.0 respectively. To keep organized the libraries and dependencies required, it has been created an environment in anaconda 2.1.1. The model described in this work as well as the ones described in the Experiments (section 3.3) have been created and trained using a workstation with the following characteristics: Windows 10 OS, Core i7, 64 Gb RAM, 2 Tb SSD, 4 processors Intel Zeon Gold 6248R and one graphic card NVIDIA Quadro RTX 6000.

2.2.1 Face Detector by means of Facial Landmarks

There are several ready-to-use real-time high-performance face detectors, such as [13] and [21] that are able to run on mobile devices. But, since this work aims not only to detect the gaze but also possible drowsiness. Then, the selected solution needs to allow to accomplish both purposes without the need of extra devices, which would mean an extra cost, and also without adding too

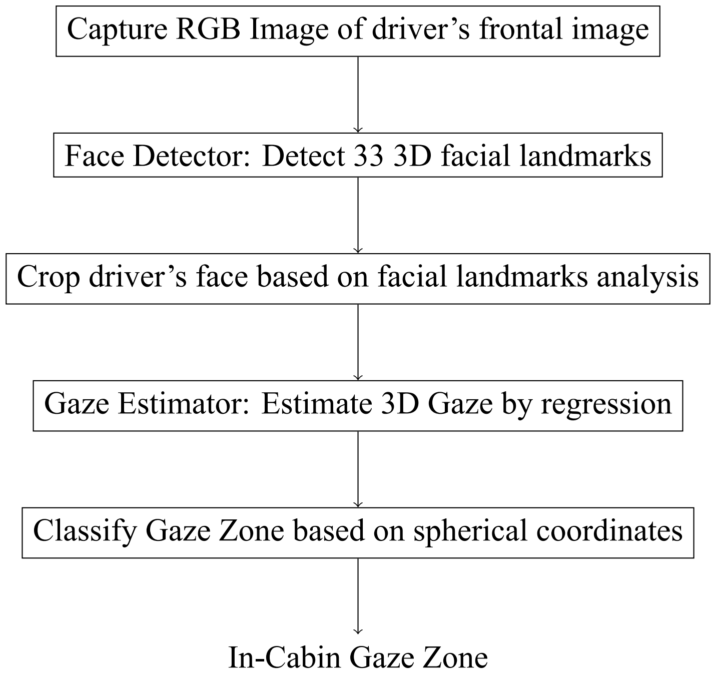

Capture RGB Image of driver’s frontal image
Face Detector: Detect 33 3D facial landmarks
Crop driver’s face based on facial landmarks analysis
Gaze Estimator: Estimate 3D Gaze by regression
Classify Gaze Zone based on spherical coordinates
In-Cabin Gaze Zone

Figure 2.3: Flowchart of the proposed system

much extra computation.

In order to detect drowsiness, it can be adopted different approaches. This thesis proposes to analyze the blink rate as well as the open and close state of the eye, both based on the eye aspect ratio (EAR). A similar approach has been followed by [48] and [50], high-performance real-time drowsiness detectors .

Since it is needed to detect the driver’s eyes as well as the face to perform the gaze estimation, then it has been used a single 3D face landmarks detector to tackle both needs. MediaPipe Face Mesh open-source Google API [27] [37] is based on a deep neural network face detector, BlazeNet [5].

MediaPipe is able to estimate 468 3D face landmarks in real-time on mobile devices and with only monocular images, it means without any additional depth sensor [27].

Starting from 2.2.1 there will be found relevant details about the model, set-up and the face detection method adopted by this thesis.

Model

The face mesh model that detects the 3D face landmarks is a deep neural network that receives as input a face detected by [5], another light-weight deep neural network. It is worth to mention that the face image has been centered, rotated and aligned in a way that a horizontal line can connect the center of eyes, that are at the same time the center of a segment the connect eyes corners. Finally, an image resize is applied, the size depends of the selected model, 256 for the full model and 118 for the smallest. It is worth to mention that the final image has 25% margin on each side [29].

Even though there is no a very detailed explanation of the model neither in the original paper [5] nor on the website [27], it can be summarized from [29] that the backbone implemented is a customized MobileNetV2-like neural network with an aggressive sub-sampling in the first layers [5]. Based on [28] it can be added that the model has multi-heads that output: 3D landmarks coordinates, 2D semantic contours and a face presence flag. It allows to perform 2D supervision and refined detection confidence

Although there is not depth information in the input, the model is able to estimate 3D face landmarks by working with the coordinates x and y of the vertices in the image pixel coordinates. The depth indicated by the coordinate z is of synthetic nature and is calculated as the depth relative to a plane passing through the mesh's center [37]. In order to keep a fixed aspect ratio between the span of x and z, the values are re-scaled proportionally to the face width [29].

MediaPipe face mesh solution is available in three different models: full, light and lightest that can be easily selected in the set-up explained in 2.2.1. The performance will be improved by increasing the model's size, but it will also result in longer inference time as described in Figure 2.1

Model(input)	IOP MAD	Time[ms] (iPhone XS)	Time[ms] (Pixel 3)
Full (256×256)	3.96%	2.5	7.4
Light (128×128)	5.15%	1	3.4
Lightest (128×128)	5.29%	0.7	2.6

Table 2.1: Performance of the Facial Landmark Detector on two mobile devices[86]

Limitations

As detailed in [29], the model will not may be able to detect face landmarks if:

- the subject is looking away from the camera more than 80[°].
- the subject is positioned too far away the camera.
- the vertical inclination is greater than 8[°].
- there is more than 50% occluded face.

The aforementioned limitations, though, do not affect the performance of the pipeline proposed by this work since result in a limited presence in the constraints imposed by the set-up of the camera detailed in 2.2.5 and the limited movements performed while driving.

Face Detection

The proposed system aims to detect the 3D gaze and the in-cabin gaze zone using a live video given by a RGB camera. It has been created a main script that reads each frame of the video, apply a pre-processing to finally detect the 3D face landmark using MediaPipe API.

To read each frame at time τ it has been used the function VideoStream of the library imutils [65]. The pre-processing converts the frame from BGR to RGB and resize it to 640,480, both operations have been performed using the library OpenCV [7].

In order to detect 3D face landmarks, it has been instantiated a MediaPipe face mesh solution and the parameters have been set-up as follows:

- *static_image_mode* = False to avoid a new landmark detection for each frame, allowing only a face tracking. It reduces latency.
- *refine_landmarks*=True to refine landmarks around eyes and lips.
- *max_num_faces*=1 represents the maximum number of faces to be detected.
- *min_tracking_confidence*=0.5 is the minimum confidence used by the face landmark detector to consider a landmark tracking as valid, otherwise a new facial detection will be automatically invoked. Higher values will increase both the performance and latency.
- *min_detection_confidence*=0.5 is the minimum confidence used by the face detector to consider a detection as valid .

Once the MediaPipe face mesh solution has been instantiated and the parameters have been set-up, the input frame is sent by reference to the solution. After the processing, the facial landmarks are returned. Sending the frame by reference by means of setting the frame as read-only before the processing, will definitely improve the performance.

It is worth to mention that the processing of the frame, to obtain the facial landmarks, may results in an empty list. It can be possible if no person (face) was detected in front of the camera or if it has been incurred in one of the limitations described in 2.2.1.

Head Orientation

Given the model limitations 2.2.1 it may result in low accurate face landmarks when the driver is looking backward. Therefore, it will may result in poor face detection as well as poor gaze estimation. This thesis proposes to tackle this problem, by only analyzing the head orientation.

If the face landmark detection results not empty, it will be returned 463 3D landmarks. Each landmark is composed by the coordinates x, y and z and have been normalized to [0.0, 1.0] by the image width and height respectively. The coordinate z represents the distance (depth) of the head's center and the camera and it uses roughly the same scale as x.

The landmarks have been analyzed to create a geometric method to detect the head's rotation angles and finally its orientation (right, left, up, down). It has been achieved by analyzing the 2D and 3D coordinates of specific parts of the face as: eyes, nose and mouth.

To come up with the head rotation and orientation, it has been needed to use the *1)* intrinsic camera parameters that allows us to link the pixel coordinates of an image point (2D) with their corresponding 3D coordinates in the world reference frame $\begin{pmatrix} 1*w & 0 & h/2 \\ 0 & 1*w & w/2 \\ 0 & 0 & 1 \end{pmatrix}$ where w and h are the frame width and height respectively and *2* the rotation matrix calculated with the functions *solvePnP* and *Rodrigues* of the library OpenCV. Finally, with the function *RQDecomp3x3* of OpenCV it has been applied a RQ decomposition to the 3x3 rotation matrix in order to get the euler angles of each coordinate x, y and z.

To detect the head orientation between right, left, up, down and backward, the euler angles has been thresholded. The values for the thresholds have been set based on personal experiments. They may need an initial calibration if there are changes in the set-up or environment.

Face detection

In order to detect the face all the face landmarks were analysed. The minimum and maximum values for the x and y coordinates were taken apart.

Thus, the four most salient points of the face mesh were found. x_{min} and x_{max} represent the beginning and the end respectively of the face mesh along the horizontal axis while y_{min} and y_{max} the beginning and the end along the vertical.

The bounding box containing the face was calculated adding a certain margin to each one of the above mentioned points. The best value of margin, covering the full head, was found to be 30% of h and 30% of w for y and x respectively. Where h is the face mesh's height and w the face mesh's width, both calculated following the Equations in 2.4.

$$\begin{aligned} h &= y_{max} - y_{min} \\ x &= x_{max} - x_{min} \end{aligned} \tag{2.4}$$

It is worth to say that the aforementioned value of margin was found by running experiments on the full gaze estimation pipeline. So, the combination of head pose and gaze vector is how the gaze zone is detected in natural driving as also mentioned in [78].

In fact, it has been seen in the experiments carried for this thesis, the fact of analysing also the head pose, allows to estimate a correct gaze zone even if the eyes are not visible enough. An example of this situation is shown in Figure 2.5.

2.2.2 Drowsiness detection

Drowsiness or fatigue is one of the main factors in road accidents [62]. Therefore, this thesis aims to detect possible drowsiness by analyzing the state of the eyes over time.

Though most recent works on drowsiness detection [17] [62] are based on machine learning and deep learning models, this thesis proposes the analysis of eyes open and closed as well as blink rate over time as performed in [56].

This has been selected mainly because of the low computational addition, robustness and good performance shown in [72] [12] [43].

The 468 3D face landmark points composing the face mesh includes 16 landmarks for each eye that represent its inner contour, as shown in 2.4.

The eye aspect ratio (EAR) is proposed by the Czech Technical University [72] [12] and has been obtained by dividing the vertical and the horizontal euclidean distance as mentioned in [43] as seen in 2.5.

Figure 2.4 shows the points selected to create the vertical (*159* and *146*) and the horizontal line (*33* and *13*).

Since the blink is usually performed by both eyes synchronously, a compounded EAR has been calculated by averaging the EAR of each eye as shown in 2.6.

When closing eyes the EAR_{lr} tends to go from its maximum value to zero. Thereby, this thesis proposes to detect drowsiness when it is detected a constant value ≤ 0.2 of EAR_{lr} in 10 consecutive number of frames. Both values may be calibrated, since the EAR_{lr} will might vary with the kind of eyes.

$$\text{EAR} = \frac{\sqrt{(x_{159}^2 - x_{145}^2) + (y_{159}^2 - y_{145}^2)}}{\sqrt{(x_{33}^2 - x_{13}^2) + (y_{33}^2 - y_{13}^2)}} \tag{2.5}$$

$$\text{EAR}_{lr} = \frac{EAR_l + EAR_r}{2} \tag{2.6}$$

2.2.3 3D Gaze Estimator

The gaze estimator aims to identify the 3D driver's gaze through a deep-learning-based model. As mentioned in the previous sections, the gaze identification will be performed on a RGB image after detection and respective

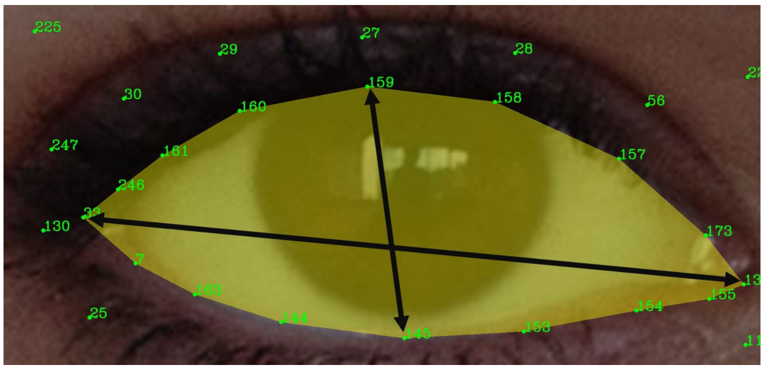


Figure 2.4: Eye landmarks returned by MediaPipe Face Mesh [27]

cropping of the driver's face.

The model created in this work is based on a deep convolutional neural network pre-trained on ImageNet [20] and fine-tuned on Gaze 360 dataset [38]. For training, basic techniques of data augmentation such as *RandomResizedCrop* and *Resize* have been adopted.

Before coming up with the proposed model, it has been created and evaluated different models. They can be grouped in:

- 1) Spatial-Temporal: Predict the gaze on a set of images by analyzing the changes along the time. Their architectures are based on spatial-temporal convolutional neural networks.

- 2) Static: Predict the gaze on a single image. Their architectures are based on convolutional neural networks and simple regressors.

The explanation of each one of the experiments performed in this thesis can be found in Section 3.3. It includes results and comparison of different architectures resulting at combining feature extractors, heads of regression and hyper-parameter set-ups. Thus, the one with the best trade-off performance-computational cost has been chosen as the proposed model of this work, and it will be the one described in the next sections.

Data Preparation

Data preparation includes the most relevant information about how the datasets and dataloaders were created, the techniques of data augmentation adopted as well as the set-up of hyper-parameters resulting in the best performance.

Dataset

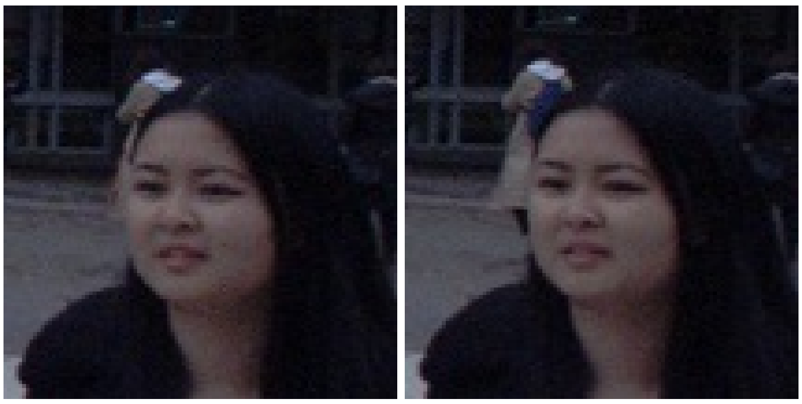
The dataset Gaze 360 is organized into 79 folders each containing two inner folders, one with full body images and the latter with head-only crops. Within each folder, there are other folders each containing a single sequence of images of the participants. In this work, only the images corresponding to the head crops have been used.

Regarding to the gaze ground truth, a .TXT file has been provided for each split (training, validation, test). Each file contains by rows the image path and the 3D coordinates of the gaze. Each of the four elements is separated by blanks.

Therefore, it has been created a custom Pytorch dataset [57] that returns a pair of tensors of type *torch.FloatTensor*. In case of working with a static model, they correspond to the image and its corresponding normalized gaze in spherical coordinates (θ, ϕ). The shapes of the tensors as proceeds: Image tensor (3,224,224) considering the image's number of channels first and later the image's shape, Gaze tensor (2) for storing (θ, ϕ).

If working with a spatial-temporal model, the dataset will always return two *torch.FloatTensor*. The first represents a sequence images to be analyzed. The length of the sequence has been set in a variable *w_size*. Therefore, the tensor shape for the sequence of images will be $(3 * w\_size, 224, 224)$ and (2) for the spherical coordinates of the gaze corresponding to the last image of the sequence.

Data Augmentation Data augmentation involves a suite of label-preserving techniques applied to deep learning models to improve the training at the same



(a) $Gaze = (0.47, -0.23, -0.85)$ (b) $Gaze = (0.41, -0.26, -0.88)$

Figure 2.5: Gaze Estimation in difficult scenarios. Whereas a) and b) appear to be the same image, their gaze vectors are different. Then, the head pose helps to distinguish where she is looking.

time avoid overfitting. It has been converted in a powerful and almost mandatory tool in the development of robust high-performance models [68].

In the proposed model, simple data augmentation techniques available in *torchvision.transforms* have been applied. The transforms have been selected by following the recommendation of [38] and [14].

The training split has been transformed only applying *RandomResizedCrop*. It basically makes random crops to the original image covering a scale or a ratio received as parameter. It has been used a scale of **0.8 and 1** to cover the 80% and 100% of the original image respectively. Finally, each crop has been resized to **[224, 244]**.

It has been shown that face processing with this initial resolution allows to handle difficult scenarios, similar to those in Figure 2.5, where the eyes are hardly visible and the gaze can be easily misestimated. There are different factors that can create these difficult scenarios, such as: wearing glasses, light reflection, dark environments, eyes' occlusion, eyes' shape etc. In Figure 2.5 those factors are eye shape and eye size.

Dataloader

To create the dataloaders of each split (train, validation and test) it has been used the Pytorch class *torch.utils.data.DataLoader*. The configuration as follows:

- dataset: the custom dataset created in Section 2.2.3.
- batch_size: the batch's size may influence the learning process, a batch's size equal to 80 gave the best performance to the proposed model.
- shuffle: True for all the splits and experiments.
- sampler: only when tuning hyper-parameters. It was used the 80% of random samples of the training.
- num_workers: 0 in all the experiments.
- pin_memory: True in all the experiments.

Model

Backbone

The gaze estimator proposed in this work uses as backbone MobileNetV2 [66] pre-trained on ImageNet and fine-tuned on Gaze360 dataset. MobileNetV2 has shown to be a good fit because of its efficiency and performance when developing on mobile and resource constrained environments [67][2].

MobileNetV2 is an improvement of MobileNetV1 and it includes a novel module in its architecture. The inverted residual blocks with linear bottleneck work with a low-dimension input that have been firstly expanded and filtered by a 3x3 convolution layer and a lightweight depth-wise separable convolution respectively. Applying this module results in a significant reduction in the parameters and floating-point operations per second (FLOPS). Which represents at the same time a reduction in the memory footprint and main memory access, common bottlenecks when developing on the edge [69] [47].

As shown in Figure 2.9 the first layer of MobileNetV2 results in a reduction of the input resolution by the half. It has been done by applying a convolution layer with 32 filters and stride of 2. Then, the activations pass trough 19 residual bottleneck layers with expansion factor of 6 except the first block that uses one of 1.

Not like its previous version, MobileNetV2 uses ReLU6 as activation function. It is a ReLU with an upper cutoff of 6 and can be thought as having 6 replicated bias-shifted Bernoulli units rather than an infinite value [42]. ReLU6 has shown robustness when working with low-precision.

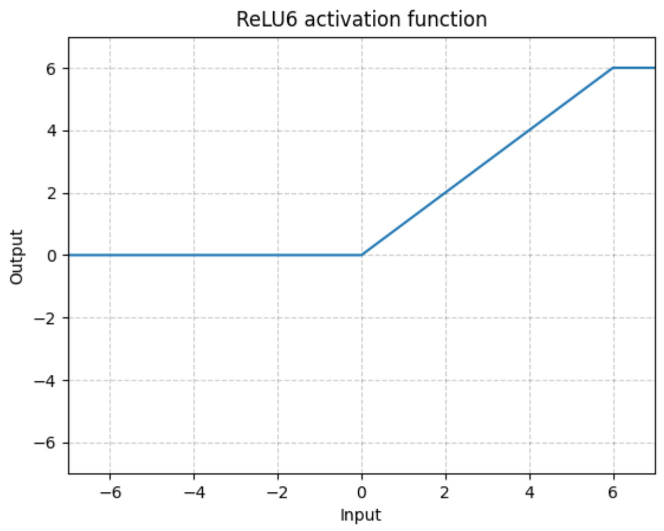


Figure 2.6: ReLU6 activation function of MobileNetV2 [22]

Some relevant parts of the MobileNetV2 architecture will be described in the next sections.

Depth-wise Separable Convolutions It is the core of the MobileNets family and most modern efficient neural network architectures. It consists on splitting the common convolution into two layers, the depht-wise and the point-wise. As can be seen in Figure 2.7, the depth-wise convolution performs a spatial filtering to each input channel (depth) by means of a single *3x3* convolution. On the other part, the point-wise performs a linear combination to the depth-wise outputs by means of *1x1* convolutions. This mix of filtering and combination results in a significant reduction of computation as well as model size.

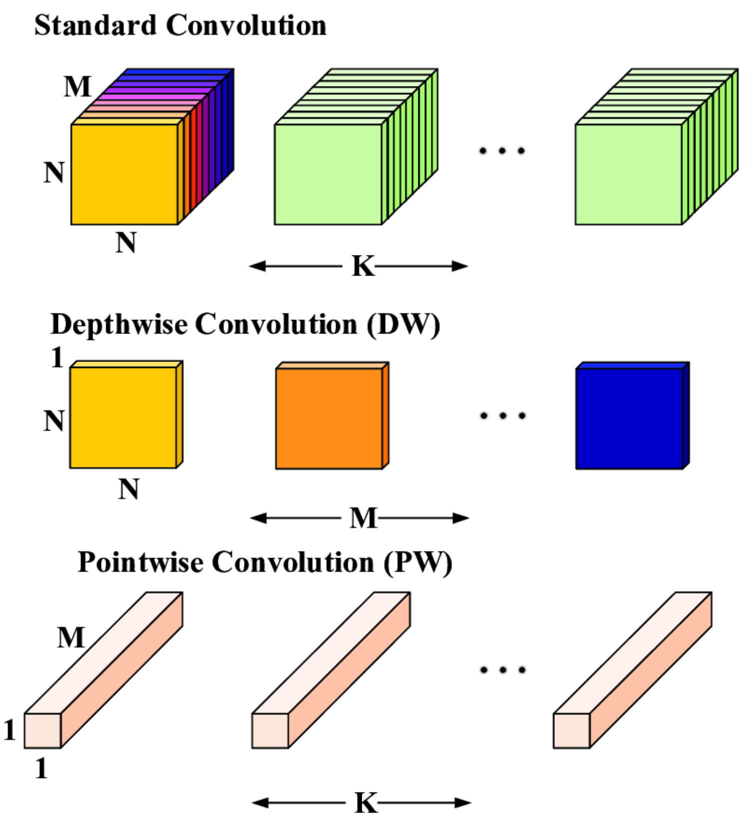


Figure 2.7: Depth-wise separable convolution introduced by MobileNets [33]

Considering an input tensor I of size C_{in} *x* W *x* H where C_{in} represents the number of channels (depth) and W, H the weight and height respectively. Applying a standard convolutional kernel of size C_{out} *x* C_{in} *x* k *x* k to produce an output O will result in a computational cost of W *x* H *x* C_{in} *x* C_{out} *x* k^2. Instead, applying a depth-wise separable convolution will result in a computational cost of W *x* H *x* C_{in} *x* *(* C_{out} + k^2 *)*. It will result in a reduction of computational cost equal to:

$$\frac{C_{out} + k^2}{C_{out} \times k^2} \tag{2.7}$$

With a reduction of almost k^2. The computation of MobileNetV2 will be of almost **9** smaller than a standard convolution since it uses a **k=3** depth-wise separable convolution [66].

Linear Bottleneck It has been shown in [66], that applying non-linear layers in bottlenecks results in a considerable reduction of performance given that ReLU squashes too much information when the features are in low dimension. Then, linear bottleneck expands the channels dimension, projecting the low dimension features into a higher dimension given an expansion ratio.

Inverted Residual Block MobileNetV2 is an updated version of MobileNetV1 and the use of Inverted Residual Blocks is part of the new changes. Known also as MBConv Blocks, they are adopted by mobile architectures that use inverted structure in exchange for efficiency. The main difference from the traditional residual block can be seen in Figure 2.8. It basically follows the narrow - wide - narrow patterns instead of wide - narrow - wide adopted by a common residual block.

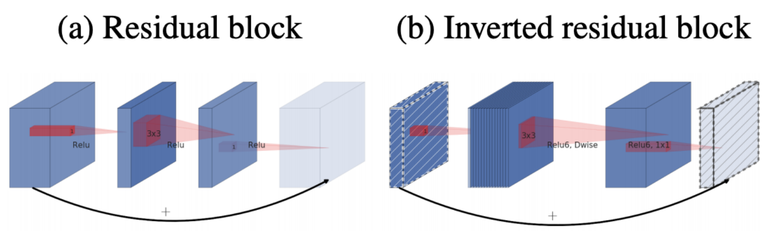


Figure 2.8: a) Traditional residual block and b) Inverted Residual Block proposed on MobileNetV2. Figure from [66]

Regression Head

Since it has been used MobileNetV2 pre-trained on ImageNet, it comes with an attached head of classification as shown below:

Backbone's output
nnDropout 0.2
nnLinear(self.last_channel, num_classes)

Table 2.2: MobileNetV2 Classification Head

The classification head has been used to build the proposed model, but the last linear layer (Table 2.2) has been modified to re-project the 1280 features of the last channel (backbone's output) into a lower dimension (256). Later, it has been attached a non-linear function (RELU).

Finally, a regression head has been attached by means of a linear layer. This layer re-projects the 256 embedding features to the required number of outputs. To estimate the gaze, the proposed model yields 3 values: θ, ϕ and confidence.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 2.9: Architecture of MobileNetV2 as backbone of the proposed model [66]

Training Set-up

The proposed model is composed by a regression head able to estimate the 3D gaze vector and confidence with a single image, and as backbone (feature extractor), a pre-trained MobileNetV2 on ImageNet, which has been fine-tuned on Gaze360 dataset on almost all the layers, except the first four. It has been used Pytorch 1.11.0 [57] and Adam optimizer.

Pytorch allows to use Multi-process data loading by setting a positive number of workers, but in this work the number of workers was set to zero. Indeed, the data was loaded on CPU and pushed it to GPU while training. To speed up the host to device transfer, it has been enabled *pin_memory* that allows the dataloader to allocate in page-locked memory. The batch size was set to 80.

It was used an initial learning rate 1×10^-3 for training the head of regression and the two last layers of the backbone (originally created as classification head), the rest of the backbone was let frozen. The training has been done for 50 epochs using ReduceLROnPlateau as learning rate scheduler. The scheduler has been set up with patience equal 2, factor equal 0.1, threshold equal 0.1, mode equal 'min' and min_lr equal $1x10^-7$.

The backbone (MobileNetV2) has been fine-tuned at all except the first four layers (initial convolution and the three bottlenecks) (see Fig. 2.9). It has been trained for 100 epochs using the same learning rate scheduler configuration used to train the head. Each one of the layer has been unfrozen in a progressive way using a decreasing learning rate that has started at 1×10^-3 and ended at 2×10^-6.

TensorboardX [73] has been used as tool of visualization. It was helpful for the progressive unfreezing of the layers. It was noticed that when unfreezing each layer per time, using the configuration of the hyper-parameters already described, the model took around ten epochs to converge. It means, that after ten epochs a new layer was unfrozen.

Loss Function

Taking into account the nature of the Gaze360 dataset, the wide range of gaze directions and head orientation in an unconstrained environment make it more difficult to estimate a precise 3D gaze vector. As indicated in [38], accuracy is difficult to achieve in this type of problem because of key aspects such as: partial or complete occlusion of one or both eyes and when the eye is viewed from a lateral angle.

Pinball loss, also known as quantile loss and hinge loss, has been used as the regression loss of the proposed model. As can be seen in Figure 2.10, its shape is similar to the trajectory of a ball on a pinball machine, hence its name. The function is always positive, and the lower the pinball loss, the more accurate the quantile prediction. In this way, the assumption that the variables behave the same in the upper tails of the distribution is eliminated [19].

Therefore, the uncertainty quantification (UQ) in the regression has been applied to model the error bounds. Quantiles 10 and 90 were used along with the mean value. Considering the output of the model (θ, ϕ, σ), the quantiles were created as shown in the table 2.3:

Quantile	Composed by
10 %	$(\theta - \sigma)$, $(\phi - \sigma)$
90 %	$(\theta + \sigma)$, $(\phi + \sigma)$

Table 2.3: Pinball Loss Quantiles

With the spherical coordinates (θ, ϕ) of the predicted gaze and ground truth, together with the confidence σ, the pinball loss (L_τ) for the quantile τ has been calculated as follows:

$$q_\tau = \begin{cases} \theta_{GT} - (\theta - \sigma), & \text{if } \tau \leq 0.5 \\ \theta_{GT} - (\theta + \sigma), & \text{otherwise} \end{cases} \tag{2.8}$$

$$L_\tau(\theta, \sigma, \theta_{GT}) = \max(t.q_\tau, -(1-t)q_\tau) \tag{2.9}$$

Where q_τ represents the predicted quantile and τ can take two values: 0.1 and 0.9. The same equations 2.8 and 2.9 have been used with the angle ϕ.

Finally the losses for both angles θ, ϕ and quantiles τ=0.1 and τ=0.9 have been averaged.

Although Pinball loss was used to train the model, angular error is the metric commonly used to validate the performance of a gaze estimator. Therefore, in Figures 2.11a and 2.11b, the behavior of angular error during training and validation can be observed.

The largest decrease in angular error was experienced during training of

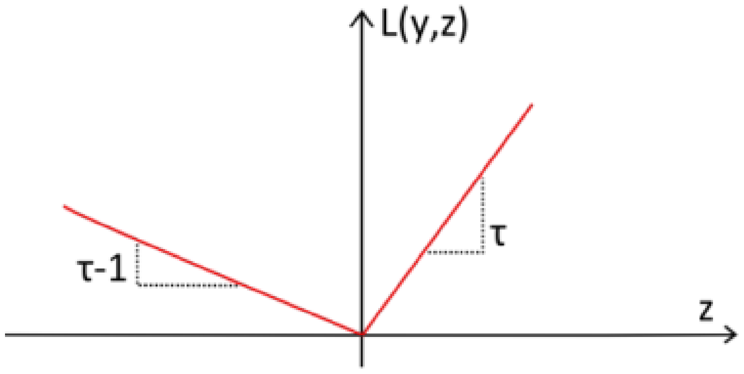


Figure 2.10: Pinball Loss

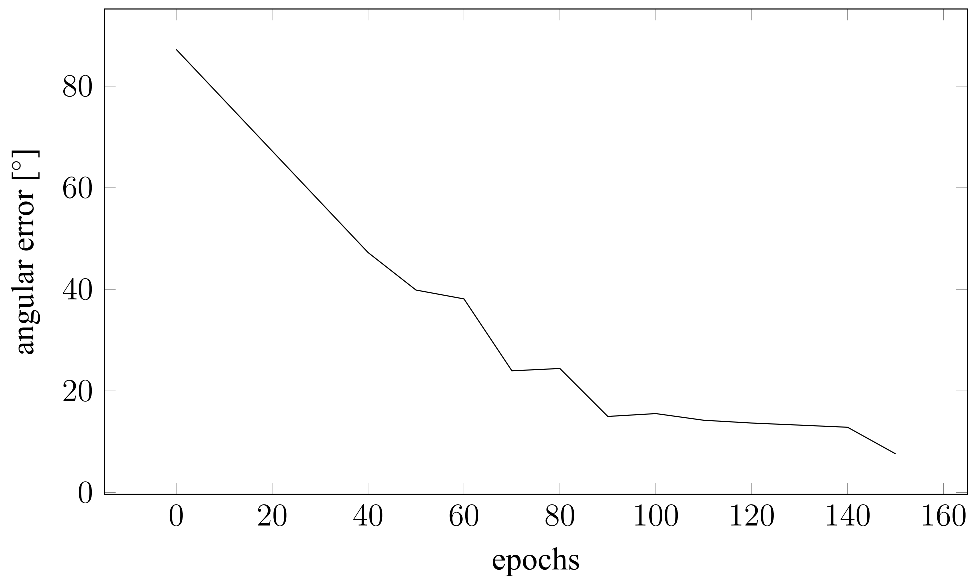
the regression head, composed of a set of linear layers, and keeping the backbone frozen. It decreased from 87.23[°] to 39.86[°] in 50 epochs. A drop rate of almost one degree per epoch (0.95[°/epoch]) was obtained.

Gaze Zone

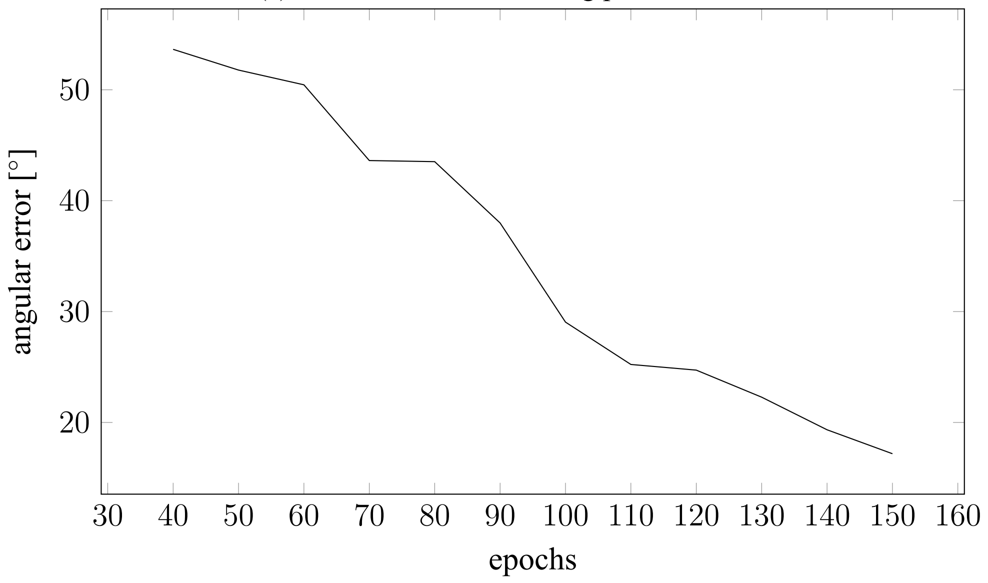
The in-cabin gaze zone is the component or area of the vehicle where the driver looks. In this work, nine gaze zones were classified according to the spherical gaze coordinates (θ, ϕ). The in-cabin zones and their descriptions are shown in Fig 2.12 and Table 2.4, respectively. They were created inspired by [18] and [78].

The workflow of the proposed system allows the gaze zones to be changed. However, an initial calibration will be required to create the angular ranges (θ, ϕ) for each zone. More technical details on calibration can be found in 2.2.5.

The process of gaze discretization and subsequent zone retrieval is straightforward and does not require training. The cabin has been divided into nine non-overlapping 2D quadrants as shown in 2.13 where each quadrant represents a zone. Yaw (θ) is placed on the horizontal axis while pitch (ϕ) is placed on the vertical axis. Thus, after having calculated the spherical coordinates of the gaze (θ, ϕ), they have been sent as a parameter to a function that returns the in-cabin zone (quadrant) where both angles are contained.



(a) Gaze Estimation Training plot curve



(b) Gaze Estimation Validation plot curve

In case of a new calibration has been performed and new angles ranges have been detected for each zone, they need to be modified in the aforementioned function.

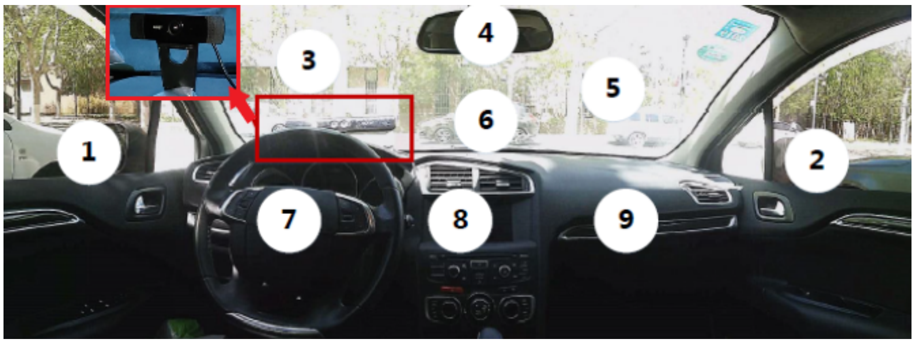


Figure 2.12: Split of the cabin into Gaze Zones

Gaze Zone Id	Description
1	Left Mirror
2	Right Mirror
3	Straight
4	Rear-View Mirror
5	Left Windshield
6	Middle Windshield
7	Steering Wheel
8	Radio
9	Glove Box

Table 2.4: Gaze zone description

2.2.4 Deployment on the edge

The pipeline usually followed when deploying on the edge can be split in two macro steps. The first involves the creation, training and evaluation of the model up to have achieved the required goals and performance, the latter relies on the deployment on device. Since deep-learning neural networks are both compute and memory intensive, thus before moving to a such limited device, it will be needed to compress the model in order to reduce the computational load. While compressing the model it will might be considered trade-offs between the computational load and latency with performance.

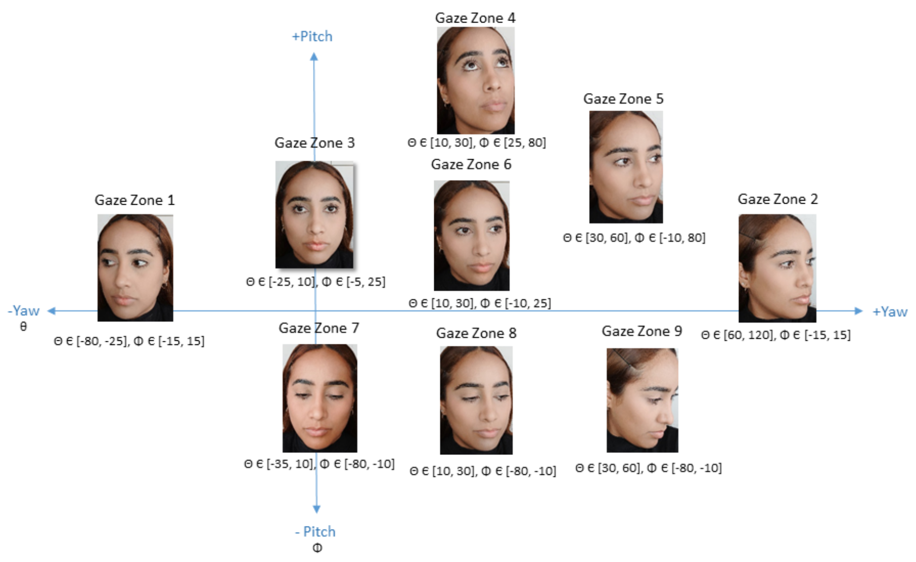

+Pitch
Gaze Zone 4
ϴ ϵ [10, 30], Φ ϵ [25, 80]
Gaze Zone 5
ϴ ϵ [30, 60], Φ ϵ [-10, 80]
Gaze Zone 3
ϴ ϵ [-25, 10], Φ ϵ [-5, 25]
Gaze Zone 6
ϴ ϵ [10, 30], Φ ϵ [-10, 25]
Gaze Zone 1
ϴ ϵ [-80, -25], Φ ϵ [-15, 15]
Gaze Zone 2
ϴ ϵ [60, 120], Φ ϵ [-15, 15]
-Yaw
θ
+Yaw
Gaze Zone 7
ϴ ϵ [-35, 10], Φ ϵ [-80, -10]
Gaze Zone 8
ϴ ϵ [10, 30], Φ ϵ [-80, -10]
Gaze Zone 9
ϴ ϵ [30, 60], Φ ϵ [-80, -10]
- Pitch
Φ

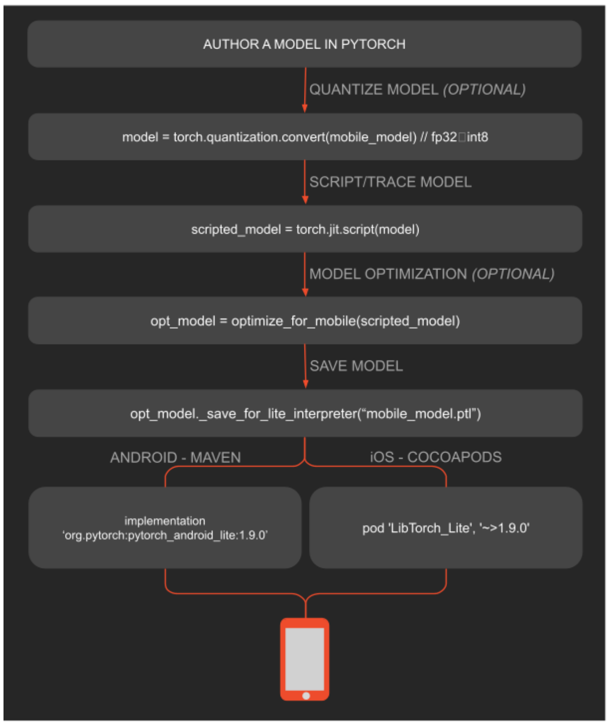
Figure 2.13: Split of the cabin into quadrants. It is based on Yaw (θ) and Pitch (ϕ) angles

Considering the device's architecture and the preferred deep learning framework, it can be chosen between a set of model optimization toolkits that allow to optimize models with the best practices and minimum complexity. Some of them are: Tensorflow Lite[74], Pytorch Mobile [61], MXNet [52] and X-Cube-AI [80]. Moreover, the deep-learning framework used to create the model should not restrict the one will be used on device. Indeed, there are open-source tools as ONNX [55] that allow to interchange models between a selection of frameworks. ONNX has been used for the experiments performed in this work, more details can be found in the section of experiments 3.3.

Regarding the development of the system proposed on this thesis, it has been used Pytorch for training and evaluating the model as well as for optimizing it (with Pytorch Mobile). The pipeline followed for deploying the model on device has been shown in Figure 2.14.

Model Optimization

Deploying deep learning Neural Networks on the edge demands to have a light-weight, memory-efficient and high-performance model. There are several techniques that allow us to speed-up by means of optimization. It can be


AUTHOR A MODEL IN PYTORCH
QUANTIZE MODEL *(OPTIONAL)*
SCRIPT/TRACE MODEL
MODEL OPTIMIZATION *(OPTIONAL)*
SAVE MODEL
ANDROID - MAVEN
iOS - COCOAPODS

```
model = torch.quantization.convert(mobile_model) // fp32□int8
scripted_model = torch.jit.script(model)
opt_model = optimize_for_mobile(scripted_model)
opt_model._save_for_lite_interpreter(“mobile_model.ptl”)
implementation
'org.pytorch:pytorch_android_lite:1.9.0'
pod 'LibTorch_Lite', '~>1.9.0'
```

Figure 2.14: Deployment on the edge pipeline adopted by Pytorch Mobile [61]

mentioned: quantization, layers fusion, pruning, distillation, clustering and scripting.

Regardless the technique chosen, the optimization of a model reduces the model’s size, latency and memory usage at the expense of some performance (accuracy, loss).

Latency is considered a crucial metric when deploying a deep learning model even worse if it will be performed on the edge. It refers to the amount of time a single sample takes to be inferred (forward pass trough the model). Being related to the power consumption of the model and entire system, so, most of time mobile deep-learning-based models strive to get low latency to even real-time inference.

In order to achieve high accuracy and low latency when estimating the 3D gaze on such resource-constrained device as a Raspberry Pi4, and according to the performance obtained, the model proposed on this work has been optimized by means of quantization and layers fusion.

Quantization

Training a deep-learning model demands propagating the loss signal trough several layers in order to update all of the parameters, which implies a lot of multiply and accumulate operations. By default, most of the Deep Learning frameworks and GPUs, represent the weights, activations and gradients as 32-bit floating numbers (FP32). Given the intense computational load of a deep-learning-based model, such high-precision representation will allow to avoid overflow issues and sometimes to converge faster. However, high-precision also means a slower and less memory-efficient model.

Recently works [40] [34] [79] have shown that most of deep learning use cases, do not need such large precision. Indeed, it rarely will be required too large magnitude (8 bits in fp32) (Figure 2.15) and in some cases the same accuracy as FP32 can be obtained with a reduced-precision data type (FP16, 16/8/4/2-bit integers).

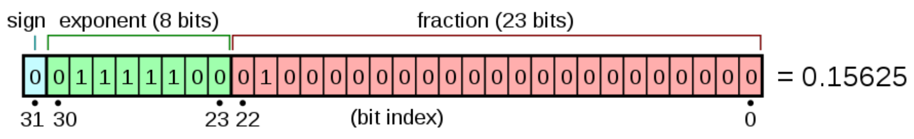


Figure 2.15: IEEE 32-bit floating point representation [70]

Quantization is a set of techniques applied to reduce basically the latency by means of transforming the floating points used by the model into integers. Considering INT8 numerical representation has a quarter as many bits of FP32, then it will be almost 4x faster. However, it is constrained to be applied only to the forward-pass.

Pytorch Mobile allows us to choose between several quantization options depending where and how the quantization will be performed.

Based on where the quantization will be applied, it can be chosen between:

- Post Training Quantization: Quantization is applied after training and the quantized parameters are learned with an initial calibration. It can be grouped into Dynamic and Static. While first refers to a dynamic during inference quantization of activations, the last refers to a fuse of activations into preceding layers together a calibration.

- Quantization Aware Training: Quantization is applied during training where the quantized parameters are also learned.

Based on how the quantization will be applied, it can be chosen between:

- Only Weight Quantization: Statically quantization for weights.
- Dynamic Quantization: Dynamic quantization for activations and statically for weights.
- Static Quantization: Statically quantization for both weights and activations.

To come up with a quantized model, it is also needed to make use of quantization backend engine (qegine). Pytorch allows to choose between the following two options:

- Qnnpack: Useful when the device is an ARM CPU.
- Fbgemm: Useful when the device is a x86 CPU with AVX2 support or higher.

On this thesis it was tried several kinds of quantization (detailed in section 3.3), the one resulting with the best performance was a 8-bit integer Static Quantization Aware Training (QAT). Even if there are faster approaches that allow to convert an already trained model into a low-precision one, the QAT models the quantization error in both the forward and backward pass using fake-quantization modules that simulate the quantize and dequantize operations in the training. A fake-quantization module has been instantiated by using the pytorch class *torch.quantization.FakeQuantize*.

It has been created a new model *GazeQuantized* that includes inside its forward pass the quantization of the input *quant(input)* and the dequantization of the output *dequant(output)* using the pytorch functions *QuantStub* and *DeQuantStub* respectively.

Layer Fusion

Another optimization technique that has been applied to the proposed model is layer fusion. It combines similar convolutional and attention layer weights to achieve higher computational efficiency [54]. As seen in Figure 2.16, the layer fusion has been applied prior to model training and quantization. Considering the backbone of the proposed model, MobileNetV2, the fusion will be applied sequentially only to the convolutional layers of the Inverted Residual Bottleneck as well as to the first three layers of Convolution + Batch Normalization + RELU located at the beginning of the model.

At the end, the quantized model resulted with fewer layers than its FP32 version, thus improving computational efficiency. In order to preserve information across the fussed layers, training was performed on both the FP32 model and the quantized model.

The below function *fuse_model* has been created to come up with the aforementioned fusion of layers.

```

from torch.quantization import fuse_modules

def fuse_model(self):
    '''
    fuse_model: fuse specific layers of the model
    '''
    for m in self.modules():
        if type(m) == ConvBNReLU:
            fuse_modules(m, ['0', '1', '2'], inplace=True
                )
        if type(m) == InvertedResidual:
                for idx in range(len(m.conv)):
                    if type(m.conv[idx]) == nn.Conv2d:
                        fuse_modules(m.conv, [str(idx)\
                        , str(idx + 1)], inplace=True)
```

Later, the backend quantization engine was configured along with the quantization settings. In this case, taking into account the architecture of the machine used for training (x86), the *fgemm* engine was chosen. Subsequently, the model was prepared and converted to a quantized version using the *prepare_qat* and *convert* functions of the *torch.quantization* library. At this point, the size of the model has been reduced by 4x from its FP32 version.

The model has been trained with the same hyper−parameter settings and data used in the FP32 version. It has converged over 10 epochs using an initial learning rate of $1 \times 10^{-}5$ and a final decaying learning rate of $1 \times 10^{-}6$ set by the scheduler *ReduceLROnPlateau*.

The quantization (QAT) workflow along with the layer fusion used in the proposed model can be seen in Figure 2.16.

From Eager to Script

Finally, the model has been transformed into a script. The Pytorch Script [59] functions convert modules and functions into a production-ready version using just-in-time compilation (JIT). This allows a faster running at non expenses of performance and without having to worry about Python run-time and python Global Interpreter Lock (GIL).

When talking about converting a model into a script, specifically when using Pytorch, the model transitions between two modes. As seen in Figure 2.17, the eager mode involves the model created so far, then before converting it into a script. This mode, allows to have a faster prototyping, training and evaluation. Then, to switch to a script mode, Pytorch allows you to choose between two functions:

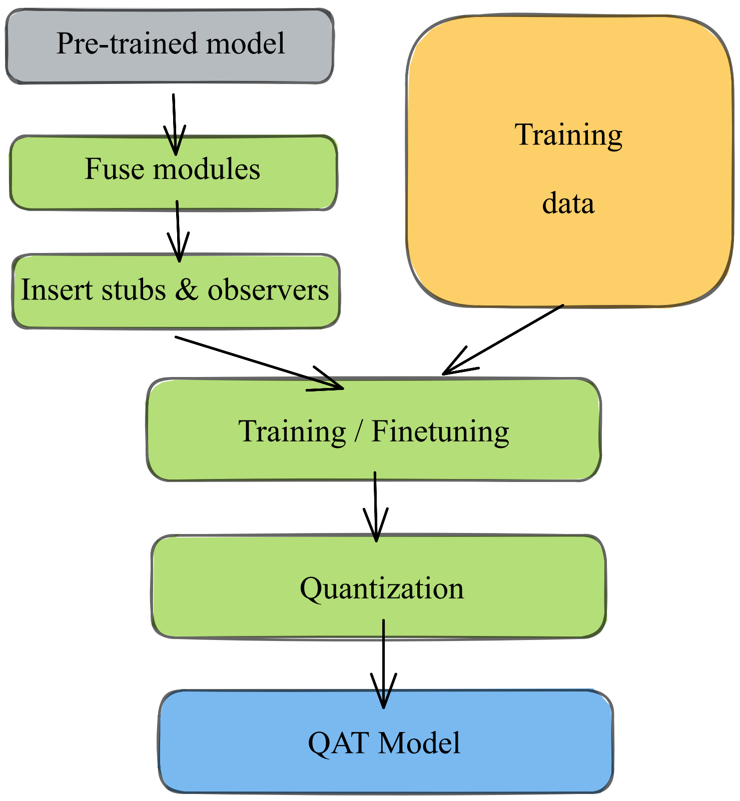


Figure 2.16: Quantization Aware Training (QAT) Pipeline

$$\begin{tikzcd}
\boxed{\text{Eager Mode}} \arrow[rrr, "\text{torch.jit.script()}", "\text{torch.jit.trace()}"'] & & & \boxed{\text{Script Mode}}
\end{tikzcd}$$

Figure 2.17: Pytorch transition from Eager to Script

- Pytorch Jit Script: It will inspect the source code, compile it as TorchScript code using the TorchScript compiler, and return a ScriptModule or ScriptFunction [59].

- Pytorch Jit Trace: Trace a function and return an executable or ScriptFunction that will be optimized using just-in-time compilation. [60].

In this work, the eager model has been converted into pytorch script using *torch.jit.script()*. Pytorch allows to perform the aforementioned transformation in a user-friendly way, as seen below:

```

from torch.jit import save, script
```

```

path = '.models/optimized/GazeMobileScript.pth'

'''
quantized_model in eager mode after QAT and layers fusion
    .
Script function of torch.jit returns a script module
    using the torch script compiler.
'''

save(script(quantized_model), path)
```

Inference on device

In order to perform inferences with the optimized model using the device proposed by this work, a 1.5GHz Raspberry Pi4 64-bit Quad core Cortex-A72 (ARM v8) SoC, the backend quantization engine Qnnpack was first configured as follows:

```
torch.backends.quantized.engine = 'qnnpack'
```

After having configured the backend quantization engine, the script version of the optimized model has been loaded as shown below:

```
path = '.models/optimized/GazeMobileScript.pth'
# Load script model
model = torch.jit.load(path, map_location='cpu')
```

And the inference has been straightforward as shown follows:

```
import torch

input_image = torch.zeros(1,3,224,224)
...
input_frame = Image.fromarray(frame, 'RGB')

# Frame as tensor
```

```
    input_image[:,:,:,:] = image_normalize(transforms.
        ToTensor()(transforms.Resize((224,224)) (input_frame))
        )

    # Inference
    output, confidence = model(input_image)
```

2.2.5 Gaze Estimation in live

In this section it will be described the list of materials, how the device (Raspberry Pi4) was configured and the gaze zones created to perform a gaze estimation on a real drive.

Materials

Below, it will be found the list of materials (hardware) used to estimate the in-cabin gaze on a Raspberry Pi4.

Hardware	Quantity
Raspberry Pi4 Model B	1
Case for Raspberry Pi4 Model B	1
External 10 Gb memory	1
Webcam Full HD	1
Smartphone as screen mirroring	1
Portable Charger 5V 3A	1

Table 2.5: List of materials used to perform gaze estimation on Raspberry Pi4

A raspberry pi4 model B 8 Gb RAM (Figure 2.18) has been used as device to run the proposed system. A case is used to enclose the raspberry and facilitate the tests on the ride. An internal cooler and whilst heat-sinks to reduce the thermal throttling (Figure 2.19).

Considering the maximum voltage (5V) and current (3A) supply (15W) supported by the device [63], it has been used a portable charger (power bank) with the required specifications.

A Full HD (1080p) 2Mpx 60fps webcam has been used to record the driver's video. It has been connected to the Raspberry by USB. How the camera is positioned in the vehicle is important, then, the set-up adopted by this work is described in section 2.2.5.

Each component of the aforementioned list of hardware can be seen in the figure 2.20

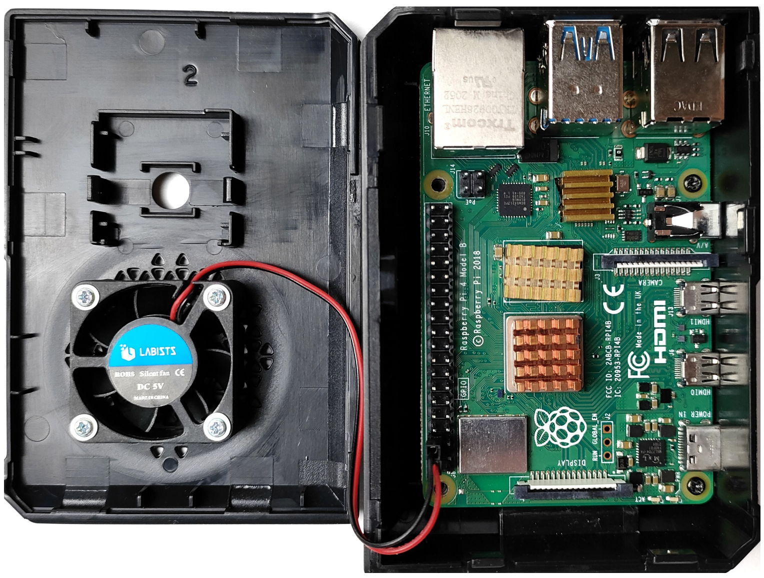


Figure 2.18: Raspberry Pi4 model B used in the experiments of this work

Set-up

In this sections it will be given more details about the set-up of raspberry Pi4 as well as how the components have been placed in the vehicle.

Raspberry Pi 4

It has been firstly upgraded the operating system to the last version, Bullseye 64bit [9]. It has been done since the available Pytorch wheels performs only on 64bit OS. Later, the value of the allowed swap memory has been incremented to 4096 Mb, in that way it can be avoided problems of segmentation

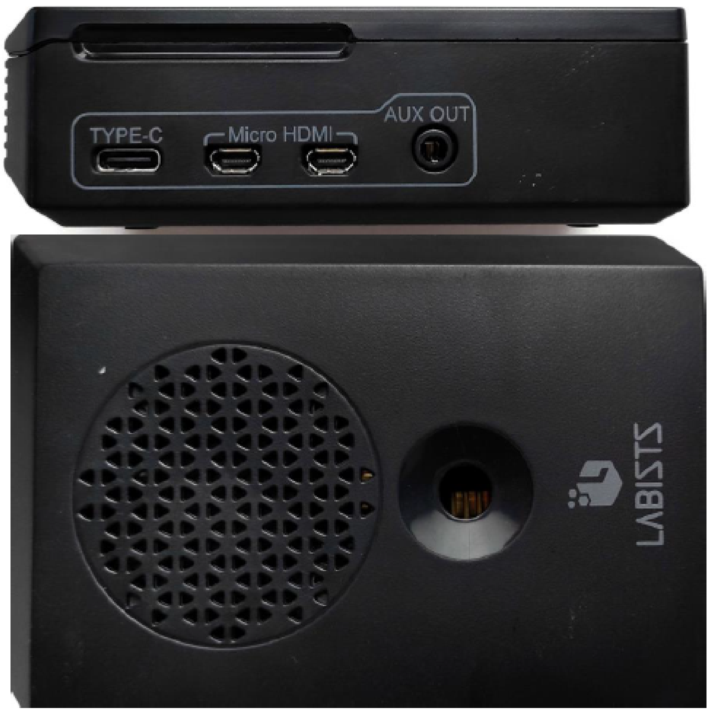


Figure 2.19: Raspberry Pi4 case

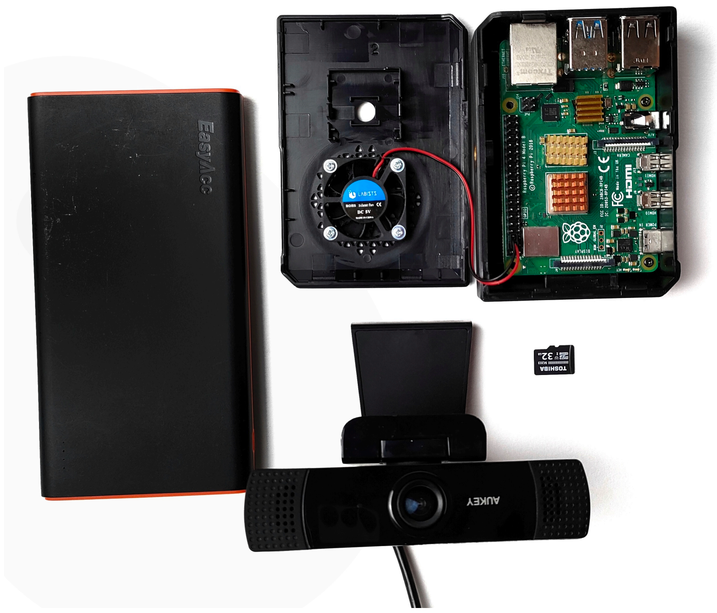


Figure 2.20: Materials used on the live demo

faults while running the system.

Regarding to the installation of the required libraries to be able to run the system, it has been downloaded mini Conda from [51] and created an environment for the project. Then, all the installs required have been done using a .YAML file, resulting from the export of the main environment (workstation used to train the model). It is worth to mention that the device has been accessed through SSH protocol, using Bitvise client [6].

To monitor the output while being on the road, it has been used a smartphone of 64 Gb RAM as screen mirroring. The connection between devices has been done using the software VNC [31]. It has been required both devices to be connected to the same network, then it has been used the smartphone to create a private wireless network (hotspot).

In-cabin set-up

The requirements regarding the set-up basically rely on how the camera is positioned. As shown in 2.21, the camera needs to be placed in front of the driver with a maximum distance of 60 cm. In no case the camera should obstruct the driver's vision and it should record the full head of the driver when he looks directly.

The embedded system, composed by the Gaze Estimator running on a Raspberry Pi4, has been enclosed in the case and looks small as shown in Figure 2.22. Before starting the ride, the system has been firstly connected to a private network hosted by the smartphone, followed by the start of the screen mirroring.

The first ride with a vehicle, which has not been used before in any test, will be used for an initial calibration of the gaze zones. More details can be found in the next section.

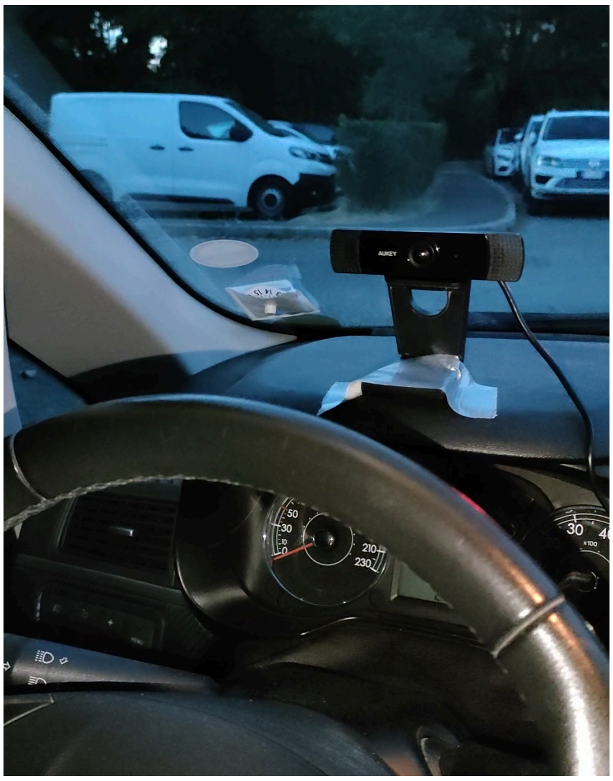


Figure 2.21: Camera set-up adopted by this work

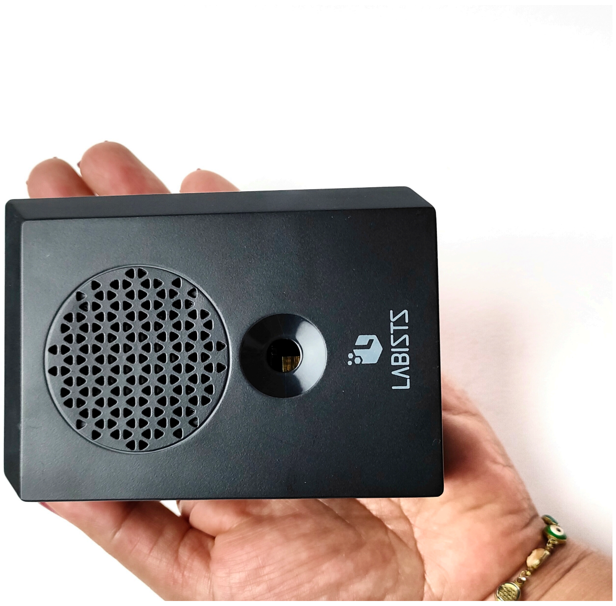


Figure 2.22: Outside view of the embedded system created in this work

2.2.6 Gaze Zones Calibration

As is mentioned in section 2.2.3, the driver's gaze zone is detected by thresholding the yaw (θ) and pitch (ϕ) of the gaze with respect to a specific setting (in-cabin 2D quadrants). Since, it may incur in different settings depending on: *1)* the dimensions of the car cabin and *2)* the position of the driver relative to the camera. Therefore, an initial calibration is necessary.

The aforementioned calibration consists on the following steps:

- 1) Creation of the gaze zones (offline).
- 2) Set-up of the camera as mentioned in 2.2.5.
- 3) Record a video of a short drive in which the driver will look at all the zones at least twice.
- 4) Extraction of frames. A pre-processing will be needed to keep only useful frames.

Later, the gaze estimator will detect the yaw (θ) and pitch (ϕ) for each frame, allowing us to create the ranges for each zone (in-cabin 2D quadrant) as seen in Figure 2.13.

Further explanation can be found in 3.2

Chapter 3

Experimental Results

3.1 Evaluation

The performance of the proposed model for estimating the 3D gaze vector of people in images has been evaluated. The images used for training and evaluation are the disjoint training and validation sets of the Gaze360 [38] dataset. Since the dataset already contained cropped heads, for evaluation it was not necessary to use the facial landmark detector described in section 2.2.1, which will allow us to focus on the main task, 3D gaze estimation. However, it has been shown in section 3.2 that the proposed model performs well even when the facial landmark detector is used as part of the pre-processing to detect and crop the heads.

The last row of Table 3.1 shows the performance of the proposed model, which has been optimized by quantization, layer fusion and JIT scripting, as described in Section 2.2.4. It performs with low latency in CPU inference time (10.83 [ms]) as well as low angular error (17.71[°]). It is worth to take note of the fact that, given the nature of the Gaze360 dataset, angles can take values up to 360[°].

When optimizing a model, regardless of the type of technique applied, the goal will be always to reduce the computational workload at the expense of performance. However, it can be seen in Table 3.1, that in our case, the drop in accuracy (angular error) from the FP32 eager version of the model to the optimized one was low. Considering the type of problem that this thesis aims to solve (in-cabin gaze zone detection), a drop of 3.3% from the initial angular error (17.15%), is not significant. Therefore, it can be said that the optimized model retains a high performance.

The small drop in performance can be explained by the applied quantization technique. Static aware training quantization (explained in section 2.2.4) is well known to maintain high accuracy while reducing computational workload.

It can also be seen in Table 3.1 how converting the eager model to a JIT script significantly reduced the memory occupancy of the parameters (almost 99.9%) and the latency in inference time (CPU) by 85.16% while maintaining almost the same accuracy reported by the quantized eager model.

Model	Parameter Memory Occupancy [Mb]	Angular Error [°]	Latency [ms]
Eager FP32	10.20	17.15	72.99
Eager INT8	2.98	17.75	62.51
JIT Script	0.01	17.72	10.83

Table 3.1: Comparison of performance of the model in their FP32 and optimized versions. **1st row:** Eager FP32 model, **2nd row:** INT8 (QAT + Layers Fusion) and the **last row:** the proposed model Pytorch JIT Script

.

In Figure 3.1 the performance of the proposed model has been compared with the top-3 3D gaze estimation architectures reported in the Gaze360 dataset benchmark [25]. It can be seen that all the models, including the proposed by this work, have similar performance (angular error can take values up 360[°]). However, the proposed model has reported a 92% reduction in parameter count and a 96% reduction in the number of operations with respect to the SOTA, and only a 7.3[°] decrease on a 360[°] scale.

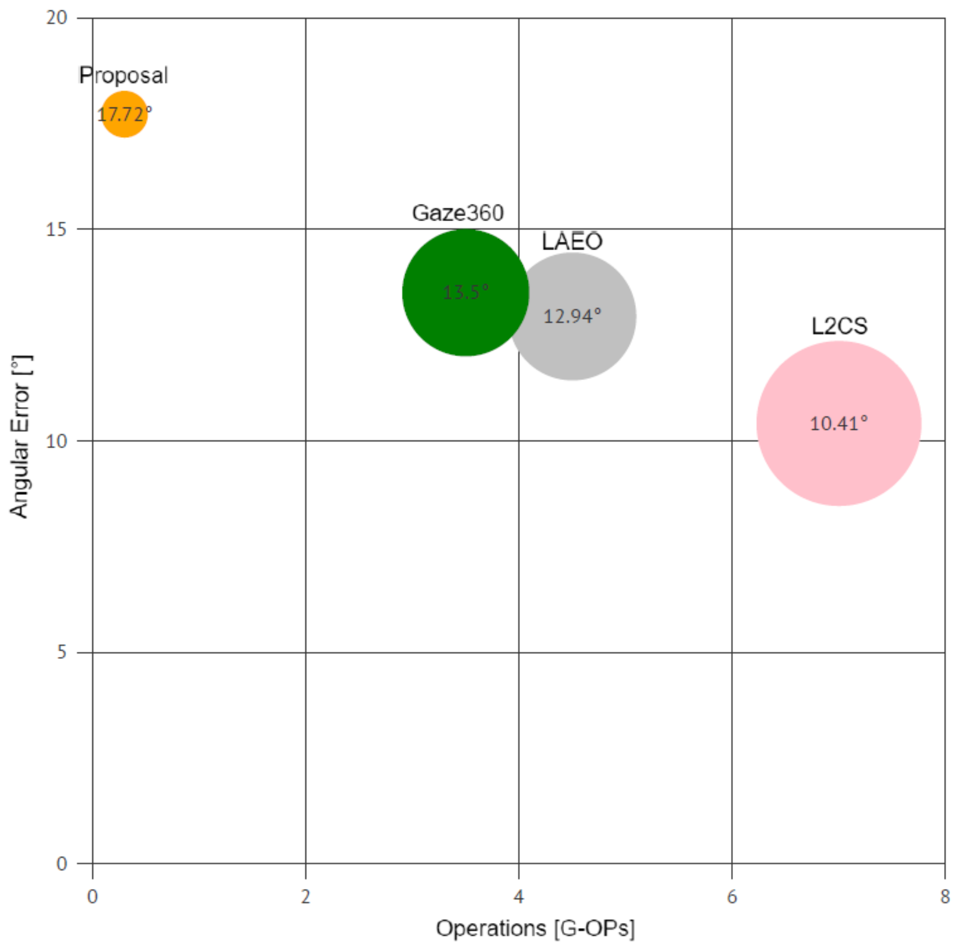


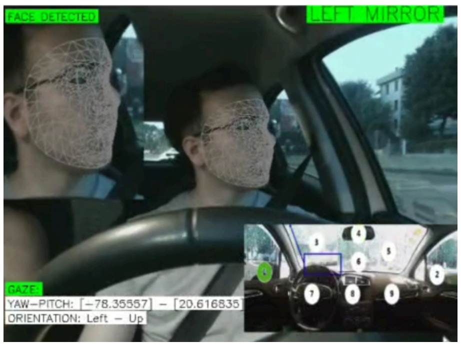
Figure 3.1: Comparison of the Proposed Model with the top-3 models reported on the Gaze360 dataset benchmark. In the x-axis the number of operations [GigaOPs]. In the y-axis the Angular Error [degree]. The size of each figure varies according to the number of parameters.

3.2 Live Demo

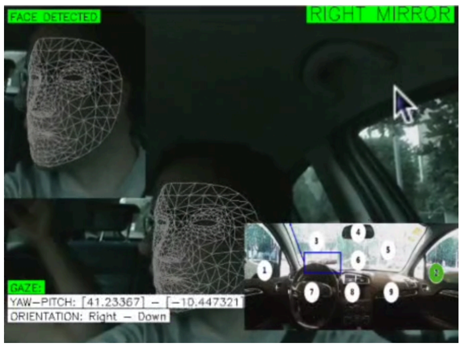
Two live demonstrations have been performed using the proposed gaze estimation system deployed on a raspberry pi 4. The first, used as an initial gaze zone calibration (see section 2.2.6), consisted on recording a video of a thirty-minute ride using the hardware list detailed in 2.2.5. The driver was previously told to look at all the nine zones in the cabin (2.12) including the backward. The video was then processed offline to extract frames and discard those repeated or invalid. Later, the 3D gaze vector and the in-cabin gaze zone were offline estimated for each frame using the system deployed on the raspberry pi. Finally, the angle ranges of each gaze zone (2D quadrant) were set in the system to be used in the next drives.

After calibration of the gaze zones, the second demonstration was performed on a thirty-minute drive in which the driver was not forced to look at all the zones, but to drive naturally. The same webcam and setup as in the first demonstration was used. However, this time, the inference was performed online by the system running on the raspberry pi 4. The 3D gaze, gaze zone and FPS information were saved in a CSV file in the local memory of the device. The data were also read in real time using a smartphone connected to the raspberry via the open source software Real VNC [31].

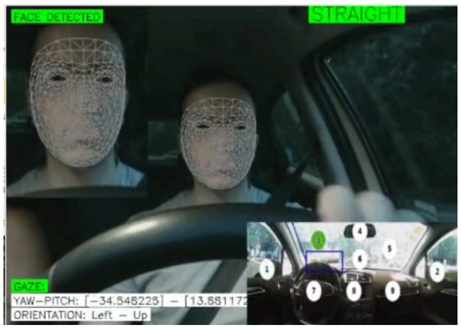
The results of the live demonstration can be seen in Figure 3.2. The images have been captured by the smartphone used as screen mirroring. In each of them, the 3D facial landmarks and the cropped face sent as input to the gaze estimation model can be seen. On the lower left, the estimated yaw and pitch angles are shown, as well as the head orientation detected by the facial landmark model. On the right side, one can see at the bottom the in-cabin gaze zones adopted in this work and at the top the gaze zone detected by the system.



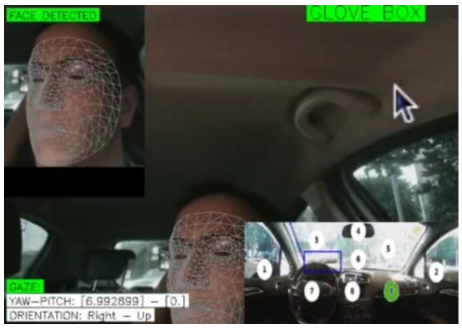
(a) Gaze zone detected: ”Left Mirror”


FACE DETECTED
RIGHT MIRROR
GAZE:
YAW–PITCH: [41.23367] – [−10.447321]
ORIENTATION: Right – Down

(b) Gaze zone detected: ”Right Mirror”



(c) Gaze zone detected: ”Straight”



(d) Gaze zone detected: ”Glove Box”

Figure 3.2: Live demonstration of the system proposed in this work. A 3D gaze estimator deployed on a Raspberry Pi4. The system is able to detect: the driver’s face and head orientation using 3D facial landmarks, the 3D gaze and nine in-cabin gaze zones.

3.3 Experiments

This section will describe each of the experiments performed in this work. The experiments cover a variation of the regression head by using three different architectures and a feature extractor different from MobileNetV2 (the one used in the proposed solution). The results of applying different techniques and engines to optimize the model are also described.

It is worth mentioning that, the feature extractor, the regression head and the optimization technique adopted by the proposed system as a solution are not explained in this section, since they have already been described in the previous sections.

Variation of Feature Extractor

Experiments have started using ResNet18 as feature extractor. It has been previously trained on ImageNet and fine-tuned on the Gaze360 dataset. As for the type of transfer learning applied, it has been with few shots. The main reason is that there was an available checkpoint [38] of ResNet18 trained on the same dataset to solve the 3D gaze estimation. As part of a technical decision in this thesis, the feature extractor was configured to return an embedding of 256 features.

Variation of Regression Head

Three models have been created, using three different regression heads: Bi-LSTM, TCN and single linear layer.

LSTM as Regression Head A bi-directional two layers LSTM has been attached to the feature extractor. Later, a linear layer with 512 ($2 \times$ number of features returned by the backbone) features as input and three (yaw, pitch and confidence) as output has been attached to the LSTM.

Since, it was the same architecture adopted by [38] and for what a checkpoint was available, then, only transfer learning with zero-shots has been performed when using ResNet18 as feature extractor.

It could be seen that although it is the same architecture and checkpoint adopted by [38], neither the validation nor the test error were similar to the one reported in the Gaze360 benchmark. The difference of about 2[°] can be explained by the variation in numerical precision of the hardware used in both works. The validation error resulted in 15.35[°] and the test error in 16.11[°].

TCN as Regression Head

A Temporal Convolutional Network (TCN) is composed by causal convolutions and dilations. Since it has more hyper-parameters than the LSTM, it has been used Optuna [3] to facilitate their tuning. Those tuned were: learning rate, optimizer, weight decay, number of layers, hidden layers and dropout of the TCN.

Given the factors of time, size of splits and resource limitation, the hyper-parameters have been tuned by taking only a subset of the training set. It has been created with random samples covering 50% of the training set. Then, the 3 best hyper-parameter settings have been selected.

Only the regression head (TCN) has been trained, and as initial checkpoint, it has been used [3]. The training has been performed for around 30 epochs using each one of the top-3 hyper-parameters settings returned by Optuna.

The best performance resulted 14.3[°] for the validation angular error. Indeed, it was the best performance obtained in this work, but, since it was also needed to consider the computational workload, the proposed model was another lighter.

It could be appreciated that, for hyper-parameter tuning, doing so on a smaller subset of the training can be useful since it will be faster than training the full set, but it is recommended to select random and representative samples. Furthermore, having trained the full set with the top-3 hyper-parameters settings gave us a higher probability of finding the best performing one.

Linear Layer as Regression Head

Since we were looking for a light-weight architecture, it has been attached a simple linear layer to perform the regression of the yaw, pitch and the confidence. In that way, it could be also seen the gap of performance by using the others head of regression. The linear layer receives as input the embedding of 256 features, which has been returned by the feature extractor, and projects it into a lower dimension of three (yaw, pitch and confidence).

The model was fine tuned with a frozen backbone for 30 epochs. It was used Adam as optimizer, 1×10^-3 as learning rate and batch size 80.

The best checkpoint resulted in 16.32[°] as validation angular error, almost 2[°] higher than when using TCN as head of regression.

Table 3.2 below shows the results obtained by each of the models mentioned above.

Model	Angular Error[°]
ResNet18 + TCN	14.67
ResNet18 + Bi-LSTM	15.35
ResNet18 + Linear Layer	16.32
MobileNetV2 + Linear Layer	17.15

Table 3.2: Performance of the different models created with ResNet18 as feature extractor and comparison with the FP32 version of the proposed model that makes use of MobileNetV2 as feature extractor .

Variation of Optimization Technique

A different technique of optimization and engine have been tried before coming up with the best-performance set-up to reduce the computational workload of the model proposed as solution in this work (MobileNetV2 as feature extractor and a linear layer as regression head).

As a different optimization engine, it has been firstly used Tensorflow Lite [74]. Since, the models have been created using Pytorch, it has been needed to convert the Pytorch model into ONNX, to finally convert it into Tensorflow. ONNX [55] comes by Open Neural Network Exchange and allows to interchange models between different Machine Learning Frameworks.

The workflow adopted can be seen in Figure 3.3 and starts with the Pytorch model already trained. Then, it is converted into ONNX by using the function *torch.onnx.export* that receives as input:

- Model: Pytorch model.
- Sample input: A tensor with the size similar to the model's input. It has been used a random tensor torch.rand((1, 7, 3, 224, 224)).
- Opset version: The ONNX opset version. In our case it was set to 12
- Input names: In our case, only one input is received, the tensor representing the driver's frontal image.
- Output names: In our case there are three outputs (θ, ϕ, σ).

Once the ONNX model has been created and then validated using the function *onnx.checker.check_model*, the model has been converted into Tensorflow by means of the functions *onnx_tf.backend.prepare* and *export_graph*. Then, the Tensorflow model has been quantized trying two post-training integer quantization techniques: 16-bit integer and 8-bit integer.

Even if both optimization techniques reduced the memory footprint of the parameters in around 4x, similar to the results obtained by the proposed solution, in both cases (16INT and 8INT), the drop of performance was significant, around 46% worse than the one reported by the eager FP32 version. Therefore, as it has been described in the quantization of the proposed solution, the optimization engine has been changed to Pytorch mobile and a quantization-aware training technique has been applied due to recommendations to avoid large drops in performance.

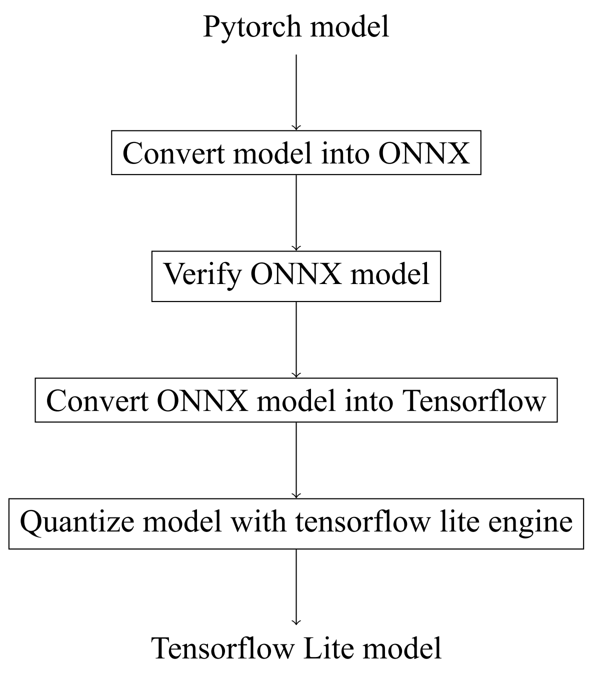

Pytorch model
Convert model into ONNX
Verify ONNX model
Convert ONNX model into Tensorflow
Quantize model with tensorflow lite engine
Tensorflow Lite model

Figure 3.3: Flowchart followed to quantize a model using Tensorflow Lite

Chapter 4

Discussion and Conclusions

4.1 Final Remarks

A lightweight, high-performance embedded system capable of detecting the 3D gaze and the in-cabin gaze zone where the driver is looking has been developed. An RGB camera has been used as the only sensor to capture the driver's frontal image.

The system is composed of two deep learning-based neural networks that work sequentially. The first is a ready-to-go face detection API, based on a MobileNet-like architecture, which analyzes the driver's frontal image and returns 3D facial landmarks. Specific landmarks have then been processed to detect the eyes and, based on their state, drowsiness could be detected. Then, parallel processing has been performed, to detect the face and crop it with an additional margin. At the end, the complete head of the driver has been cropped from the original image and sent as input to the second model. This model is in charge of estimating the 3D gaze vector using a light-weight architecture, MobileNetV2, as a backbone.

In order to run on RaspberryPi4, the eager FP32 version of the model has been optimized by the techniques: 8-bit integer quantization aware training and layers fusion, using Pytorch Mobile. Subsequently, the already optimized eager model has been converted into a JIT script, resulting in near-SOTA performances with almost 90% less memory footprint.

Finally, the 3D gaze vector has been classified as a gaze zone based on the yaw and pitch angles (θ, ϕ) and the in-cabin configuration. This configuration, basically divides the vehicle cabin into disjoint 2D quadrants, one for each zone. In this work, nine gaze zones have been proposed including: right mirror, left mirror, straight ahead, rear view mirror, center windshield, right windshield, glove box, steering wheel and radio.

4.2 Limitations

This work aims to estimate the 3D gaze vector using as input a RGB image containing the driver's frontal image. To ensure accurate results as mentioned in 3.1, the RGB image needs to contain a human head or at least the face.

Although the proposed system aims to detect the in-cabin gaze zones, those located in front of the driver facing the camera (yaw $\theta \in$ [-90,90]), it is also able to detect a backward gaze zone. This last is done by means of a head orientation analysis.

When detecting frontal in-cabin gaze zones, it will be needed a RGB image containing a full head and at least one visible eye. Even if the eyes are opened or closed, the system will be able to estimate the 3D gaze vector and the in-cabin gaze zone.

Considering the dataset used for training the neural network did not contain images of people wearing glasses, so the fact the driver wear glasses, it may impact the performance given the strong reflections and distortions created. Even it is a problem, it can mitigated by fine tuning the neural network, as explained in 4.3

4.3 Future work

Given the nature of the Gaze360 dataset, the variety of head orientation, gaze direction and environments, it can be considered one of the most challenging datasets for performing gaze estimation. Although the results obtained in this thesis have been satisfactory, it should be useful to evaluate the performance of the model on other datasets such as: MPIIGaze [85] and GazeCapture [41].

It has been noticed that images with poor light as well as the presence of glasses or dark sunglasses may incur in a drop of performance. It can result in a wrong gaze zone classification since on such scenarios, it will be difficult to detect the eyes and gaze. However, this kind of problem can be mitigated by using external sources of light, infrared sensors as well as a better camera.

It has been observed that when processing images in low light, as well as those in which the driver is wearing eyeglasses or sunglasses, where the eyes are not sufficiently visible, a drop in performance may be incurred. It may result in a wrong classification of the gaze zone, as it will be difficult to detect the eyes and gaze in such scenarios. However, this type of problem can be mitigated by using external light sources, infrared sensors as well as a better camera.

Improved drowsiness detection can be implemented by monitoring other types of signals, e.g.: frequency of mouth opening, hand and head movements, as well as analysis of other body parts useful for detecting drowsiness.

To reduce latency at the time of inference, the skip of redundant frames during streaming can be proposed, as well as the use of higher performance camera. As for model optimization, other techniques, such as pruning, can be applied.

The proposed system can be adapted to ADAS to gain insight into driver behavior. Signals such as speed and steering wheel turn can be analyzed together with the 3D gaze vector to detect distractions and future actions, alltogether to ensure a safe road. The system can also be used for different or complementary approaches to safety. For example, it can work together with an augmented reality system that, based on the 3D gaze, displays a virtual dashboard with specific options that the driver can select with his or her eyes.

Outside the automotive domain, 3D gaze can be useful for detecting customer behavior in a grocery store. PoG (2D coordinates point of gaze) can be used to detect user behavior on a website or mobile application.

Bibliography

[1] C. Ahlström, A. Anund, and E. H. Kjellman. Stress, fatigue and inattention amongst city bus drivers–an explorative study on real roads within the adas & me project. In *6th International Conference on Driver Distraction and Inattention (DDI2018)*, volume 2, pages 1–7, 2018.

[2] L. Ai, Z. Luo, C. Wang, and Y. Wu. Mobilenet investigation: its application and reproducing edge detectors using depth-wise separable convolution. In *ICMLCA 2021; 2nd International Conference on Machine Learning and Computer Application*, pages 1–6. VDE, 2021.

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. *CoRR*, abs/1907.10902, 2019. arXiv: 1907.10902. URL: http://arxiv.org/abs/1907.10902.

[4] A. A. Bamidele, K. Kamardin, N. S. N. Abd Aziz, S. M. Sam, I. S. Ahmed, A. Azizan, N. A. Bani, and H. M. Kaidi. Non-intrusive driver drowsiness detection based on face and eye tracking. *International Journal of Advanced Computer Science and Applications*, 10(7), 2019.

[5] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann. Blazeface: sub-millisecond neural face detection on mobile gpus. *arXiv preprint arXiv:1907.05047*, 2019.

[6] Bitvise SSH Client. https://www.bitvise.com/ssh-client-download/. [Online; accessed 10-May-2022].

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[8] C. Braunagel, E. Kasneci, W. Stolzmann, and W. Rosenstiel. Driver-activity recognition in the context of conditionally autonomous driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 1652–1657. IEEE, 2015.

[9] Bullseye Raspberry Pi OS. `https://www.raspberrypi.com/news/raspberry-pi-os-debian-bullseye/`. [Online; accessed 10-May-2022].

[10] D. B. Carr and P. Grover. The role of eye tracking technology in assessing older driver safety. *Geriatrics*, 5(2), 2020. ISSN: 2308-3417. DOI: `10.3390/geriatrics5020036`. URL: `https://www.mdpi.com/2308-3417/5/2/36`.

[11] R. C. Castanyer, S. Martínez-Fernández, and X. Franch. Integration of convolutional neural networks in mobile applications. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*, pages 27–34. IEEE, 2021.

[12] J. Cech and T. Soukupova. Real-time eye blink detection using facial landmarks. *Cent. Mach. Perception, Dep. Cybern. Fac. Electr. Eng. Czech Tech. Univ. Prague*:1–8, 2016.

[13] M. N. Chaudhari, M. Deshmukh, G. Ramrakhiani, and R. Parvatikar. Face detection using viola jones algorithm and neural networks. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6. IEEE, 2018.

[14] C.-S. Chen, H.-T. Lin, et al. 360-degree gaze estimation in the wild using multiple zoom scales. *arXiv preprint arXiv:2009.06924*, 2020.

[15] Z. Chen and B. E. Shi. Appearance-based gaze estimation using dilated-convolutions. In *Asian Conference on Computer Vision*, pages 309–324. Springer, 2018.

[16] Y. Cheng, H. Wang, Y. Bao, and F. Lu. Appearance-based gaze estimation with deep learning: a review and benchmark. *arXiv preprint arXiv:2104.12668*, 2021.

[17] V. R. R. Chirra, S. R. Uyyala, and V. K. K. Kolli. Deep cnn: a machine learning approach for driver drowsiness detection based on eye state. *Rev. d'Intelligence Artif.*, 33(6):461–466, 2019.

[18] I.-H. Choi, S. K. Hong, and Y.-G. Kim. Real-time categorization of driver's gaze zone using the deep learning techniques. In *2016 International conference on big data and smart computing (BigComp)*, pages 143–148. IEEE, 2016.

[19] Y. Chung, W. Neiswanger, I. Char, and J. Schneider. Beyond pinball loss: quantile methods for calibrated uncertainty quantification. *CoRR*, abs/2011.09588, 2020. arXiv: `2011.09588`. URL: `https://arxiv.org/abs/2011.09588`.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: a large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[21] C. N. Duong, K. G. Quach, I. Jalata, N. Le, and K. Luu. Mobiface: a lightweight deep learning face recognition on mobile devices. In *2019 IEEE 10th international conference on biometrics theory, applications and systems (BTAS)*, pages 1–6. IEEE, 2019.

[22] H. et al. Pytorch ReLU6 activation function. `https://pytorch.org/docs/stable/generated/torch.nn.ReLU6.html/`, 2017. [Online; accessed 10-May-2022].

[23] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu. Rmpe: regional multi-person pose estimation. In *Proceedings of the IEEE international conference on computer vision*, pages 2334–2343, 2017.

[24] K. A. Funes Mora, F. Monay, and J.-M. Odobez. Eyediap: a database for the development and evaluation of gaze estimation algorithms from rgb and rgb-d cameras. In *Proceedings of the symposium on eye tracking research and applications*, pages 255–258, 2014.

[25] Gaze360 Benchmark. https://paperswithcode.com/sota/gaze-estimation-on-gaze360/. [Online; accessed 10-May-2022].

[26] S. Gillet, R. Cumbal, A. Pereira, J. Lopes, O. Engwall, and I. Leite. Robot gaze can mediate participation imbalance in groups with different skill levels. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, pages 303–311, 2021.

[27] Google. MediaPipe Face Mesh. https://google.github.io/mediapipe/solutions/face_mesh/, 2019. [Online; accessed 10-May-2022].

[28] Google. MediaPipe Face Mesh CVPR 2019 Official Slides. https://sites.google.com/view/perception-cv4arvr/facemesh/, 2019. [Online; accessed 10-May-2022].

[29] Google. MediaPipe Face Mesh Model Card. https://drive.google.com/file/d/1QvwWNfFoweGVjsXF3DXzcrCnz-mx-Lha/preview/, 2019. [Online; accessed 10-May-2022].

[30] K. Harezlak and P. Kasprowski. Application of eye tracking in medicine: a survey, research issues and challenges. *Computerized Medical Imaging and Graphics*, 65:176–190, 2018.

[31] A. Harter. Real VNC. https://www.realvnc.com/es/connect/download/viewer/raspberrypi/, 2010. [Online; accessed 10-May-2022].

[32] J. He, K. Pham, N. Valliappan, P. Xu, C. Roberts, D. Lagun, and V. Navalpakkam. On-device few-shot personalization for real-time gaze estimation. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0, 2019.

[33] 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. *CoRR*, abs/1704.04861, 2017. arXiv: `1704.04861`. URL: `http://arxiv.org/abs/1704.04861`.

[34] A. Jain, S. Bhattacharya, M. Masuda, V. Sharma, and Y. Wang. Efficient execution of quantized deep learning models: A compiler approach. *CoRR*, abs/2006.10226, 2020. arXiv: `2006.10226`. URL: `https://arxiv.org/abs/2006.10226`.

[35] S. Kapp, M. Barz, S. Mukhametov, D. Sonntag, and J. Kuhn. Arett: augmented reality eye tracking toolkit for head mounted displays. *Sensors*, 21(6):2234, 2021.

[36] A. Kar and P. Corcoran. A review and analysis of eye-gaze estimation systems, algorithms and performance evaluation methods in consumer platforms. *IEEE Access*, 5:16495–16519, 2017.

[37] Y. Kartynnik, A. Ablavatski, I. Grishchenko, and M. Grundmann. Real-time facial surface geometry from monocular video on mobile gpus. *arXiv preprint arXiv:1907.06724*, 2019.

[38] P. Kellnhofer, A. Recasens, S. Stent, W. Matusik, and A. Torralba. Gaze360: physically unconstrained gaze estimation in the wild. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6912–6921, 2019.

[39] M. Q. Khan and S. Lee. Gaze and eye tracking: techniques and applications in adas. *Sensors*, 19(24):5540, 2019.

[40] D. S. Khudia, J. Huang, P. Basu, S. Deng, H. Liu, J. Park, and M. Smelyanskiy. FBGEMM: enabling high-performance low-precision deep learning inference. *CoRR*, abs/2101.05615, 2021. arXiv: 2101.05615. URL: https://arxiv.org/abs/2101.05615.

[41] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik, and A. Torralba. Eye tracking for everyone. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2176–2184, 2016.

[42] A. Krizhevsky and G. Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010.

[43] A. Kuwahara, K. Nishikawa, R. Hirakawa, H. Kawano, and Y. Nakatoh. Eye fatigue estimation using blink detection based on eye aspect ratio mapping (earm). *Cognitive Robotics*, 2:50–59, 2022.

[44] A. S. Le, T. Suzuki, and H. Aoki. Evaluating driver cognitive distraction by eye tracking: from simulator to driving. *Transportation research interdisciplinary perspectives*, 4:100087, 2020.

[45] J.-h. Lee, I. Yanusik, Y. Choi, B. Kang, C. Hwang, J. Park, D. Nam, and S. Hong. Automotive augmented reality 3d head-up display based on light-field rendering with eye-tracking. *Optics Express*, 28(20):29788–29804, 2020.

[46] J. Z. Lim, J. Mountstephens, and J. Teo. Emotion recognition using eye-tracking: taxonomy, review and current challenges. *Sensors*, 20(8):2384, 2020.

[47] D. Liu, H. Kong, X. Luo, W. Liu, and R. Subramaniam. Bringing ai to edge: from deep learning's perspective. *Neurocomputing*, 2021.

[48] C. B. S. Maior, M. J. das Chagas Moura, J. M. M. Santana, and I. D. Lins. Real-time classification for autonomous drowsiness detection using eye aspect ratio. *Expert Systems with Applications*, 158:113505, 2020.

[49] S. Mehta and M. Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *CoRR*, abs/2110.02178, 2021. arXiv: 2110.02178. URL: https://arxiv.org/abs/2110.02178.

[50] S. Mehta, S. Dadhich, S. Gumber, and A. Jadhav Bhatt. Real-time driver drowsiness detection system using eye aspect ratio and eye closure ratio. In *Proceedings of international conference on sustainable computing in science, technology and management (SUSCOM), Amity University Rajasthan, Jaipur-India*, 2019.

[51] Mini Conda. https://github.com/conda-forge/miniforge/. [Online; accessed 10-May-2022].

[52] MXNet. https://mxnet.apache.org/versions/1.9.1/. [Online; accessed 10-May-2022].

[53] R. A. Naqvi, M. Arsalan, G. Batchuluun, H. S. Yoon, and K. R. Park. Deep learning-based gaze detection system for automobile drivers using a nir camera sensor. *Sensors*, 18(2):456, 2018.

[54] J. O'Neill, G. V. Steeg, and A. Galstyan. Compressing deep neural networks via layer fusion. *CoRR*, abs/2007.14917, 2020. arXiv: 2007.14917. URL: https://arxiv.org/abs/2007.14917.

[55] ONNX. https://onnx.ai/about.html/. [Online; accessed 10-May-2022].

[56] N. N. Pandey and N. B. Muppalaneni. Real-time drowsiness identification based on eye state analysis. In *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, pages 1182–1187. IEEE, 2021.

[57] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: an imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. arXiv: 1912.01703. URL: http://arxiv.org/abs/1912.01703.

[58] S. Peißl, C. D. Wickens, and R. Baruah. Eye-tracking measures in aviation: a selective literature review. *The International Journal of Aerospace Psychology*, 28(3-4):98–112, 2018.

[59] Pytorch Jit Script. https://pytorch.org/docs/stable/generated/torch.jit.script.html/. [Online; accessed 10-May-2022].

[60] Pytorch Jit Trace. https://pytorch.org/docs/stable/generated/torch.jit.trace.html/. [Online; accessed 10-May-2022].

[61] Pytorch Mobile. https://pytorch.org/mobile/home/. [Online; accessed 10-May-2022].

[62] M. Ramzan, H. U. Khan, S. M. Awan, A. Ismail, M. Ilyas, and A. Mahmood. A survey on state-of-the-art drowsiness detection techniques. *IEEE Access*, 7:61904–61919, 2019.

[63] Raspberry Pi4 Model B Specifications. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. [Online; accessed 10-May-2022].

[64] A. Recasens, A. Khosla, C. Vondrick, and A. Torralba. Where are they looking? *Advances in neural information processing systems*, 28, 2015.

[65] A. Rosebrock. Imutils. https://pypi.org/project/imutils/, 2021. [Online; accessed 10-May-2022].

[66] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381.

[67] S. Savitz, C. Perera, and O. Rana. Edge analytics on resource constrained devices. *International Journal of Computational Science and Engineering*, 2021.

[68] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[69] J. Shuja, K. Bilal, W. Alasmary, H. Sinky, and E. Alanazi. Applying machine learning techniques for caching in next-generation edge networks: a comprehensive survey. *Journal of Network and Computer Applications*, 181:103005, 2021.

[70] Single-precision$_floating - point_format$. https://en.wikipedia.org/wiki/Single-precision_floating-point_format/. [Online; accessed 10-May-2022].

[71] B. A. Smith, Q. Yin, S. K. Feiner, and S. K. Nayar. Gaze locking: passive eye contact detection for human-object interaction. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 271–280, 2013.

[72] T. Soukupova and J. Cech. Eye blink detection using facial landmarks. In *21st computer vision winter workshop, Rimske Toplice, Slovenia*, 2016.

[73] TensorboardX. https://tensorboardx.readthedocs.io/en/latest/tutorial.html/. [Online; accessed 10-May-2022].

[74] Tensorflow lite. https://tensorflow.google.org/lite/. [Online; accessed 10-May-2022].

[75] G. Wang, Z. P. Bhat, Z. Jiang, Y.-W. Chen, D. Zha, A. C. Reyes, A. Niktash, G. Ulkar, E. Okman, and X. Hu. Bed: a real-time object detection system for edge devices. *arXiv preprint arXiv:2202.07503*, 2022.

[76] J. Wang and E. Olson. Apriltag 2: efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4193–4198. IEEE, 2016.

[77] L. Wang and C. Wang. Gaze estimation of multi-camera and multi-screen system oriented to human-computer interaction. In *The International Conference on Cyber Security Intelligence and Analytics*, pages 786–792. Springer, 2022.

[78] Y. Wang, G. Yuan, Z. Mi, J. Peng, X. Ding, Z. Liang, and X. Fu. Continuous driver's gaze zone estimation using rgb-d camera. *Sensors*, 19(6):1287, 2019.

[79] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius. Integer quantization for deep learning inference: principles and empirical evaluation. *CoRR*, abs/2004.09602, 2020. arXiv: `2004.09602`. URL: `https://arxiv.org/abs/2004.09602`.

[80] XCubeAI. `https://www.st.com/en/embedded-software/x-cube-ai.html/`. [Online; accessed 10-May-2022].

[81] J. Xu, J. Min, and J. Hu. Real-time eye tracking for the assessment of driver fatigue. *Healthcare technology letters*, 5(2):54–58, 2018.

[82] R. Zhang, A. Saran, B. Liu, Y. Zhu, S. Guo, S. Niekum, D. Ballard, and M. Hayhoe. Human gaze assisted artificial intelligence: a review. In *IJCAI: Proceedings of the Conference*, volume 2020, page 4951. NIH Public Access, 2020.

[83] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: an extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083,

2017. arXiv: 1707.01083. URL: http://arxiv.org/abs/1707.01083.

[84] X. Zhang, Y. Sugano, and A. Bulling. Evaluation of appearance-based methods and implications for gaze-based applications. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–13, 2019.

[85] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. Appearance-based gaze estimation in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4511–4520, 2015.

[86] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. It's written all over your face: full-face appearance-based gaze estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 51–60, 2017.

Acknowledgements

First of all, I would like to thank my mother for her constant support and endless love. For always being patient and supportive with me, especially when I need it most, when I am in my weird informatics mood.

Thanks to my grandparents, who together with my mother, have raised me as a woman of values and principles. I am a firm believer that a child's future starts at home.

I would like to thank the University of Bologna for opening the doors to international students, supporting them and making them feel less far from home throughout their studies.

I would like to thank Prof. Francesco Conti and Prof. Samuele Salti for their constant support throughout this thesis. Thank you for having shared with me valuable insights and for having supported me at all times.

I would like to thank Gianluca Toscano, Denny Di Pardo and all my coworkers at TEORESI for being supportive, inclusive and kind throughout the internship and thesis project.

I would like to thank my boyfriend and his mother, for being my angels at all times. Thank you for motivating me to learn a new language, Italian, which definitely allowed me to enjoy my time here more. Thank you for accepting my loud laughs and last-minute plans. Thank God I met you at the right time, without them, all this would have been more difficult.

I would like to thank Joice Arcos and her family for their constant endorsement, especially during my first arrival in Italy. For being my first friends and for their support in those moments when I was not even able to say ”Salve, vorrei un caffè per favore”.

Last but not least, I would like to thank the Angely of three years ago. Thank you for being determined, adventurous and risk-taking by deciding to leave your whole life in Ecuador and cross the Atlantic to achieve your dreams, no matter that it would mean starting from scratch.

Love you all.