Video-Based Sleepiness Prediction in Naturalistic Driving Environments

Master's thesis in Data Science and AI

Casper Lindberg and Anton Claesson

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Video-Based Sleepiness Prediction in Naturalistic Driving Environments

Casper Lindberg and Anton Claesson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Signal processing and Biomedical Engineering
Computer vision and medical image analysis
Chalmers University of Technology
Gothenburg, Sweden 2021

Video-Based Sleepiness Prediction
in Naturalistic Driving Environments
Casper Lindberg and Anton Claesson

Supervisor: Tomas Björklund, Volvo Cars
Supervisor: Che-Tsung Lin, Electrical Engineering
Examiner: Christopher Zach, Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Signal processing and Biomedical engineering
Computer vision and medical image analysis
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Video-Based Sleepiness Prediction
in Naturalistic Driving Environments
Casper Lindberg, Anton Claesson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Driver sleepiness is the cause of many road accidents. This project explores two machine learning approaches for predicting a driver's sleepiness level based on infrared video data collected in a naturalistic driving environment.

The first approach fits a Random Forest sleepiness classifier on 36 handcrafted features based on eyelid movements, such as percentage of eyelid closure over time (PERCLOS). To obtain these features, a Random Forest eye state classifier was built to predict open, partially closed, or closed eyes. With this approach, the Observer Rated Sleepiness (ORS) level (0-4) of a driver could be classified with a Mean Squared Error (MSE) of 1.21 averaged over eight separate predictors fitted in a nested Leave-One-Out Cross Validation (LOOCV) study. Furthermore, we show that the number of training drivers significantly affects the ability to predict sleepiness.

The second sleepiness prediction approach is based on intensities of facial movements and micro-expressions over time, fed into a Long Short-Term Memory (LSTM) neural network. This model was unable to learn to predict sleepiness on unseen drivers, although only a few hyperparameter configurations were investigated due to long training times.

Keywords: driver sleepiness, drowsiness detection, blink detection, sleepiness prediction, naturalistic driving, driver monitoring system (DMS).

Acknowledgements

We want to thank our supervisor Tomas Björklund at Volvo Cars for invaluable guidance through this project. Without the many discussions with him, this project would not have been possible. We would also like to thank Emma Tivesten at Volvo Cars for all the insights and feedback. Of course, we also wish to thank Erik Hjerpe, who introduced us to all the great people at Volvo Cars and made this project possible. We would also like to thank our examinator Christopher Zach and supervisor, Che-Tsung Lin at Chalmers, for taking the time to help us with this project and for providing valuable feedback. Finally, a special thanks to all the exceptional people and friends at Chalmers who have made these five years the great time it has been.

Casper Lindberg, Anton Claesson - Gothenburg, June 2021

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

ANN Artificial Neural Network. 17
AU Action Unit. xvii, 14–16, 31, 48

BPTT back-propagation through time. 18

CE-CLM Convolutional Experts Constrained Local Model. xiii, 9, 11, 12
CEN Convolutional Experts Network. xiii, 11, 12
CLMs Constrained Local Models. 9, 10, 14
CLNF Constrained Local Neural Field. xiii, 14, 16
CNN Convolutional Neural Network. 9, 11, 83, 86
CV Cross Validation. xvi, 19, 48, 74

EAR Eye Aspect Ratio. 31, 48, 55, 79
EEG electroencephalography. 1, 5

FACS Facial Action Coding System. xvii, 14, 15, 48, 83
FFNN Feed Forward Neural Network. 17
FN False Negatives. 20
FP False Positives. 20
FPS frames per second. xvii, 7, 25, 26, 28, 44, 50, 51

GPU Graphics processing unit. 50

HOG Histogram of Oriented Gradients. 15, 16

IR infrared. 2, 3, 8, 9, 23, 78, 80, 85
ISO International Organization for Standardization. xiii, xvii, 6, 7

KSS Karolinska Sleepiness Scale. xiii, xvii, 5, 6, 24, 25, 34, 35

LOOCV Leave-One-Out Cross Validation. v, xiv, xv, xviii, 19, 32, 33, 35–37, 39, 46, 48–50, 55, 57, 58, 60–64, 68, 69, 71, 76, 78, 79, 85
LSTM Long Short-Term Memory. v, xiv–xvi, 18, 34, 48–51, 60, 71, 72, 74, 76, 82, 83, 85, 86

MAP maximum *a posteriori*. 10
MLP Multi-layer Perceptron. xix, 22
MSE Mean Squared Error. v, xiv–xvi, xviii, 15, 20, 35–37, 39, 40, 46, 60–66, 69–72, 74, 76, 80, 81, 85
MTCNN Multi-Task Convolutional Neural Network. 9, 11

NLLL negative log likelihood loss. 48

ORS Observer Rated Sleepiness. v, xiii, xv, xvii, 5, 6, 20, 24, 25, 34, 35, 40, 41, 46, 48, 60, 63, 65, 69–71, 76, 80, 81, 83, 85, 86

PCA Principal Component Analysis. 15, 16
PDM Point Distribution Model. 9–11, 15
PERCLOS percentage of eyelid closure over time. v, 5, 8, 41, 44, 83

RLMS Regularised Landmark Mean Shift. 10, 14
RNN Recurrent Neural Network. xiv, 17, 18, 34, 48, 49, 83

SICS Swedish Institute of Computer Science. 24
SVM Support Vector Machines. 16
SVR Support Vector Regression. 16

TCN Temporal Convolutional Network. 18, 83, 86
TN True Negatives. 20
TP True Positives. 20
TPE Tree-Structured Parzen Estimator. 21, 46, 65

1

Introduction

This section covers a brief introduction and the main goals of the project. A short introduction to the area of driver sleepiness prediction and related works is also given. Finally, limitations and a detailed specification of issues under investigation are presented.

1.1 Background

Every year, 1.35 million people die in road traffic crashes [8]. According to statistics from the U.S. Department of Transportation [9], 2.4 percent of these fatalities in the U.S. can be attributed to driver sleepiness. Another study from AAA Foundation For Traffic Safety [10] reported that in 8.8 to 9.5 percent of the crashes, observable driver drowsiness assessed based on eyelid closures was present. Detecting sleepiness makes it possible to implement in-vehicle countermeasures that could avoid crashes and save lives. In this project, the goal is to research, develop, and evaluate a driver sleepiness prediction model based on video data for Volvo Cars.

1.2 Related Works

There is no agreed-upon golden standard for measuring driver sleepiness. Thus, many previous works to detect driver drowsiness utilize different evaluation methods, metrics, data collection environments, and datasets.

Driver alertness is dependent on many different factors such as circadian rhythm, sleep quality and quantity, accumulated lack of sleep, motivation, stress, and monotony [11]. As an example, in a study by E. Vural et al. [12], they allowed the driver to fall asleep and crash multiple times during a three-hour driving simulation. They defined a drowsy state to be the minute before instances of the driver either crashing or falling asleep, while non-drowsy states were picked from the beginning of the driving simulation. To predict drowsiness, E. Vural et al. [12] utilized standardized actions of facial movements. This indicates that facial action movements could be a useful predictor in the setting of this project as well, even though data collection environments, experimental setup, and evaluation methods may differ.

Other work assess sleepiness by using physiological measurements such as electroencephalography (EEG), which measures electrical activity in the brain [13, 14].

Self-reported or observer-reported sleepiness is also common when evaluating driver drowsiness in naturalistic driving due to the simplicity of such techniques [15].

1.3 Aim

The main goal of this project is to develop and evaluate a method to assess driver drowsiness based on IR video data. In turn, this method can be applied to larger datasets to refine the detection of severe driver drowsiness further. Consequently, this method can guide the development of future advanced driver assistance systems that can reduce accidents related to severe sleepiness.

Apart from classifying the level of sleepiness, it is also of interest to determine whether the driver's eyes are closed or not. By knowing the state of the eyes, it is possible to detect microsleeps and handcrafted features related to blinks and long eye closures, which have shown to be important sleepiness indicators. Therefore, another goal is to build and evaluate a different model capable of determining the state of the driver's eyes in each video frame. With such a classifier, microsleeps and instances of sleeping could be detected even if the sleepiness level prediction is less precise. Additionally, successful results from this model could be utilized for sleepiness level prediction in a second step.

This work is not meant to result in a finished ready-to-implement sleepiness detection system. Instead, the focus lies within exploring the feasibility of the task at hand. Thus, there are a few questions which this project aims to answer to guide future work. These questions include finding a suitable video sequence duration for predicting a driver's sleepiness and estimating the amount and variation of additional data required for a generalized sleepiness prediction system. These questions are further specified in Section 1.5, Specific Research Questions.

1.4 Limitations

- The available video data is captured by a SmartEye IR camera facing the driver and another side-facing IR camera. This work will only make use of the camera facing the driver.
- The developed model is limited to consider nighttime video only, as no data was collected during daylight hours.
- Since the available data only includes eight test drivers, a generalized system for driver sleepiness prediction or eye state classification is not expected. However, the estimated generalization abilities of the systems are still of interest to estimate the amount of additional data that must be collected to generalize the model.

1.5 Specific Research Questions

This project aims to provide answers to the following research questions:

I. Can a machine learning system learn to classify the driver's sleepiness level from videos using an IR video camera facing the driver?

II. Is it possible to determine the state of the driver's eyes (opened or closed) using the same camera? Can this information be used to determine the driver's sleepiness level?

III. What duration of a video segment is required to predict the driver's level of sleepiness?

IV. How well does the sleepiness prediction models trained on video data generalize to other persons?

V. What is the estimated amount and variation of data required to build generalized versions of the eye opening/closure state and sleepiness prediction models?

VI. What are the most important visual facial movement actions when it comes to detecting sleepiness using a machine learning model regarding the dataset at hand?

2

Theory

In this chapter, measures and sleepiness indicators are defined based on existing standards. Next, the theory behind the facial feature detection methods is covered. Furthermore, to classify the eye state and driver sleepiness level, a few supervised machine learning prediction algorithms evaluated in this work are introduced. This section is followed by the principles behind training and evaluating machine learning classifiers. Finally, two data management formats used in the project are described.

2.1 Measuring Sleepiness

Sleepiness is defined as the transitional state between wakefulness and sleep and is challenging to measure [15]. Sleepiness reporting based on self-reported or physiological data can be too intrusive and affect the experiment. Other unobtrusive methods include eye- and head tracking or reporting of the Observer Rated Sleepiness (ORS), where an observer rates the sleepiness of the subject. Karolinska Sleepiness Scale (KSS) is an example of a self-reported sleepiness scale where subjects indicate which level best reflects their experience during the last ten minutes [16] or another interval suitable for the test set-up.

For the dataset used in this project, ORS is defined as the driver's sleepiness during the last five minutes, rated by an observer in the backseat of the car. The KSS level of the driver is also present in the dataset. These scales are shown in further detail in Table 2.1, where the different levels are described.

2.1.1 Common Indicators of Sleepiness

Many features based on eyelid movement has shown to be correlated with sleepiness. As an example, percentage of eyelid closure over time (PERCLOS) has previously been considered such a superior measurement that it has been used as the actual target for which some drowsiness detection systems are evaluated on [11]. However, U. Trutschel et al. [11] also argue that caution should be taken when estimating driver alertness based on PERCLOS only, as it was found that other physiological measurements such as EEG contain more accurate fatigue information. Nevertheless, features based on eyelid movement are arguably the most common to use for sleepiness detection, and in many cases, these approaches have shown promising

Table 2.1: The Karolinska Sleepiness Scale (KSS), which is estimated by the driver, and the Observer Rated Sleepiness (ORS), which is the driver's sleepiness estimated by an observer in the backseat. KSS is updated every ten minutes of driving, ORS is updated every five minutes.

Karolinska Sleepiness Scale (KSS)	
Value	**Description**
1	Extremely alert
2	Very alert
3	Alert
4	Rather alert
5	Neither alert nor sleepy
6	Some signs of sleepiness
7	Sleepy, but no effort to keep awake
8	Sleepy, but some effort to keep awake
9	Very sleepy, great effort to keep awake, fight sleepy
10	Extremely sleepy, can't keep awake

Observer Rated Sleepiness (ORS)	
Value	**Description**
0	No signs of sleepiness, normal wakefulness
1	No microsleep, light sleepiness, fighting sleep slightly, driving ability OK
2	Single/very few microsleeps, fighting sleep heavily, impaired driving ability
3	Microsleep, cannot manage to fight sleep, heavily impaired driving ability
4	Obviously falling asleep, crossing the border line, "intervention"

results.

In this section, we first define three different states of the eyes. Using these definitions, we can calculate several different features based on eyelid movement.

2.1.1.1 Eye State Definitions

Three different *eye states* are defined based on the International Organization for Standardization (ISO) standard [1]: *open eyes* ('0'), *partially closed eyes* ('1') and *closed eyes* ('2'). These are illustrated in Figure 2.1. Since the eyes can open and close individually, we choose to define the eye state of the driver in a single frame depending on the most closed eye. For example, if one eye is open and the other is partially open, the eye state of that particular image assumes a partially closed value.

2.1.1.2 Eye Blink Features

This section presents the eyelid movement-based features for detecting driver drowsiness evaluated in this project. Each feature is derived from the definition of the eye states in Section 2.1.1.1.

Based on the ISO standard [1], we define three different types of blinks which can occur at any time in a video sequence; *normal blinks*, *long blinks*, and *microsleeps*. The type of any occurring blink is determined based on the duration of the eyelid closure. The duration consists of two phases, the closing and opening phase. In the closing frame, the state of the eye goes from open to closed and vice versa for the

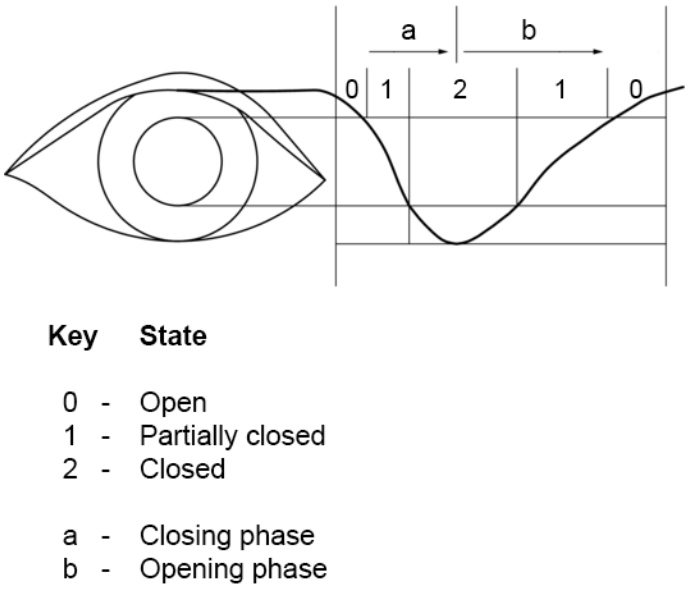

Figure 2.1: Phases of a blink, according to the ISO standard. The image has been modified to fit the terminology used in this report. Additionally, the closing and opening phases have been shortened to not include the open state.
Source: Adapted from [1].

Table 2.2: Different types of blinks defined based on the ISO standard [1]. The number of frames corresponding to the specified duration is calculated with the assumption of capturing video at 30 FPS.

Blink Type	Duration	No. of frames
Normal blinks	<=300 ms	3-9
Long blinks	300 ms to 500 ms	10-15
Microsleeps	> 500 ms	> 15

opening phase. Note that in this project, we have defined the closing phase as the duration between the open and closed eye state. The opening phase is defined as the duration between the closed and open eye state. As a result, the duration of the blink is slightly shorter than in the ISO standard [1]. In Table 2.2, the defining duration for each blink type is shown, along with the corresponding number of frames in a video sampled at 30 FPS.

We choose to distinguish between frames where the eyes close entirely and where the eyes only close partially. Therefore, the types *normal blink*, *long blink*, and *microsleep* are defined as blinks containing at least one closed eye state. On the other hand, a *half blink* is any blink that only contains partially closed eye states, such that the eyes never close completely. By making this separation, we also obtain the types *half normal blink*, *half long blink*, and *half microsleep*.

The count and duration can be measured for each of these blink types. As an example, there might be two microsleeps with a total duration of 10 seconds for a 30-

Table 2.3: Additional eye blink feature definitions which can be extracted given the eye state in a video sequence.

Eye blink feature	Definition	Unit
Max blink duration	The maximal duration of a blink	Time (ms)
Eyes closed duration	Accumulated time of the eyes being closed, state 2	Time (ms)
Eyes half-closed duration	Accumulated time of the eyes being partially closed, state 1	Time (ms)
PERCLOS	Percentage of eyelid closure, state 2, over time	Percent (%)
Half PERCLOS	Percentage of partial eyelid closure, state 1, over time	Percent (%)
Eyes closing duration	Time of closing the eye	Time (ms)
Eyes opening duration	Time of opening the eye	Time (ms)

second video sequence where the driver is tired. Thus, we can construct the features 'microsleep count' and 'microsleep duration' related to the sequence. Similarly, 'normal blink count', 'normal blink duration','long blink count', and 'long blink duration' can be calculated. We call these types of features *eye blink features*, as they are related to the blinking pattern of the driver. The features can serve as input to a machine learning model for classifying sleepiness.

Additional eye blink features acquired from the eye states are defined in Table 2.3. These are inspired by a paper by M. Dreissig et al. [17] called "*Driver Drowsiness Classification Based on Eye Blink and Head Movement Features Using the k-NN Algorithm.*" Finally, for features measured with a duration, more eye blink features can be obtained by calculating the average, median, and variance of the blink durations in the sequence.

Doing all of these calculations for the different blink types and additional eye blink features defined in Table 2.2 and Table 2.3 results in 36 different eye blink features.

2.2 Infrared Imaging

The cameras used in the driver sleepiness study were of near-infrared type to work well in low light conditions. According to a technical specification of a SmartEye camera, the wavelength is 850 nm [18]. Moreover, SmartEye's near infrared cameras "see through" glasses and sunglasses of non-IR type [19].

2.3 Facial Behavior Analysis

In order to detect sleepiness using video sequences, facial movement and behavior of the driver must be analyzed. The OpenFace 2.0: Facial Behavior Analysis

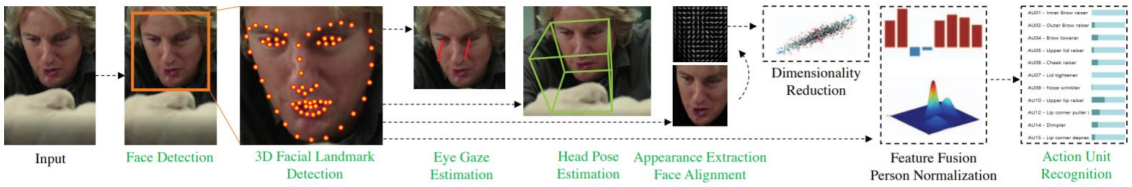

Figure 2.2: OpenFace 2.0 facial behavior analysis pipeline, including: landmark detection, head pose and eye gaze estimation, facial action unit recognition. The outputs from all of these systems (indicated in green) can be saved to disk or sent via network in real-time. Source: [2].

Toolkit [2] can be used for this purpose. It is capable of face detection, facial landmark detection, head pose estimation, facial action unit recognition, and eye-gaze estimation, see Figure 2.2. The toolkit is able to cope with non-frontal faces, occluded faces, and low illumination conditions through the use of a new Convolutional Neural Network-based face detector and a new and optimized facial landmark detection algorithm [2]. Since it is stated to work well in low illumination conditions, its potential on grey-scale near-infrared images during the night is promising.

2.3.1 Face Detection

The 3D Facial Landmark Detection model is initialized with a bounding box of the face given by the Multi-Task Convolutional Neural Network (MTCNN) model [20]. It works well in various poses, illuminations, and occlusions and is built on a deep cascaded multi-task framework. The deep cascaded multi-task framework exploits the inherent correlation between face detection and alignment. The cascaded structure consists of three stages of deep convolutional networks that predict face and landmark locations in a coarse-to-fine manner. Stage 1 produces candidate windows of faces quickly through a shallow Convolutional Neural Network (CNN) called a fast Proposal Network (P-Net). In stage 2, the candidate windows of faces are refined through a more complex CNN called a Refinement Network (R-Net). The R-Net rejects a large number of non-face windows. In the third stage, the Output Network (O-Net), which is also a CNN, produces a final bounding box [20].

2.3.2 3D Facial Landmark Detection and Tracking

OpenFace 2.0 uses an implementation of the state-of-the-art Convolutional Experts Constrained Local Model (CE-CLM) for facial landmark detection, and tracking optimized to enable real-time performance [2]. Before the CE-CLM model, Constrained Local Models (CLMs) were popular for facial landmark detection [3]. CLMs model the appearance of each facial landmark individually using local detectors and use a 3D shape model to perform constrained optimization. The local detectors make CLMs robust to occlusion. Moreover, the 3D shape model allows CLMs to handle different poses and landmark self-occlusion [3], e.g., when the person is looking to the side.

The CLMs consist of three important parts: a Point Distribution Model (PDM),

patch experts (or local detectors), and the fitting approach. The PDM determines the location of facial feature points in the image. The patch experts model the appearance of local patches around facial landmarks. An example fitting approach is the Regularised Landmark Mean Shift (RLMS). The fitting approach is used to estimate the rigid and non-rigid parameters $\mathbf{p}$, which fit the underlying image best. The fitting approach can be described by

$$\mathbf{p}^* = \arg\min_{\mathbf{p}}[\mathcal{R}(\mathbf{p}) + \sum_{i=1}^{n} \mathcal{D}_i(\mathbf{x}_i; I)] \ , \tag{2.1}$$

where $\mathcal{R}$ represents the regularisation term that penalizes overly complex or unlikely face shapes. $\mathcal{D}$ represents the misalignment of landmark i at $\mathbf{x}_i$ in the image I [21]. The 2D location of the i^{th} feature in the image can be described through the PDM as

$$\mathbf{x}_i = s \cdot R_{2D} \cdot (\bar{\mathbf{x}}_i + \mathbf{\Phi}_i \mathbf{q}) + \mathbf{t} \ , \tag{2.2}$$

where $\bar{\mathbf{x}}_i = [\bar{x}_i, \bar{y}_i, \bar{z}_i]^T$ represents the mean value of the i^{th} element of the PDM. $\mathbf{\Phi}_i$ is a $3 \times m$ principal component matrix that describes the linear variations of non-rigid shape of this feature point together with the m dimensional vector $\mathbf{q}$. The rigid shape parameters are parametrised with the scaling term s, a 2D translation $\mathbf{t} = [t_x, t_y]^T$, and an orientation $\mathbf{w} = [w_x, w_y, w_z]^T$ which controls the rotation matrix R_{2D} consisting of the first two rows of a 3×3 rotation matrix R. The whole face model shape can be described as $\mathbf{p} = [s, \mathbf{t}, \mathbf{w}, \mathbf{q}]$ [21].

The patch expert (local detector) $\pi_{\mathbf{x}_i}$ determines the probability of the i^{th} landmark being aligned at a particular image location $\mathbf{x}_i$ according to

$$\pi_{\mathbf{x}_i} = \mathcal{C}_i(\mathbf{x}; \mathcal{I}) \in [0, 1] \ . \tag{2.3}$$

The patch expert can be modeled as the output of regressor $\mathcal{C}_i$ where 1 is perfect alignment and 0 is no alignment [21].

In CLMs, the goal is to estimate the maximum *a posteriori* (MAP) probability of the face model parameters $\mathbf{p}$ given the initial estimation $\mathbf{p}_0$ [22]. The face model parameters $\mathbf{p}$ are initialized by a face detection step. In the fitting procedure, a parameter update $\Delta\mathbf{p}$ is required to get closer to the optimal solution $\mathbf{p}^*$. The fitting objective can be described, similarly to Equation 2.1, by [21],

$$\mathbf{p}^* = \arg\min_{\mathbf{p}_0 + \Delta\mathbf{p}} [\mathcal{N}(\mathbf{p}_0 + \Delta\mathbf{p}) + \sum_{i=1}^{n} \mathcal{D}_i(\mathbf{x}_i; \mathcal{I})] \ . \tag{2.4}$$

With Regularised Landmark Mean Shift, a least-squares solution can be found with the expression in [21],

$$\arg\min_{\Delta\mathbf{p}}(\|\mathbf{p}_0 + \Delta\mathbf{p}\|^2_{\Lambda^{-1}} + \|J\Delta\mathbf{p}_0 - \mathbf{v}\|^2) \ . \tag{2.5}$$

In Equation 2.5 above, J is the Jacobian of the landmark locations with respect to the parameter vector $\mathbf{p}$ evaluated at $\mathbf{p}_0$. Λ^{-1} is a matrix describing the prior

on parameter $\mathbf{p}$. A uniform distribution is used for rigid shape parameters while a Gaussian distribution prior $p(\mathbf{p}) \propto \mathcal{N}(\mathbf{q}; \mathbf{0}, \mathbf{\Lambda})$ is used for non-rigid shapes. $\mathbf{v} = [\mathbf{v}_1, ..., \mathbf{v}_n]^T$ represents the mean-shift vector over the patch responses that approximate the response map using a Gaussian Kernel Density Estimator described by

$$\mathbf{v}_i = \sum_{\mathbf{y}_i \in \Psi_i} \frac{\pi_{\mathbf{y}_i} \mathcal{N}(\mathbf{x}_i^c; \mathbf{y}_i, \rho\mathbf{I})}{\sum_{\mathbf{z_i} \in \mathbf{\Psi_i}} \pi_{\mathbf{z}_i} \mathcal{N}(\mathbf{x}_i^c; \mathbf{z}_i, \rho\mathbf{I})} - \mathbf{x}_i^c \ , \tag{2.6}$$

where ρ is an empirically determined parameter [21].

The update rule can be derived using Tikhonov regularized Gauss-Newton method:

$$\Delta\mathbf{p} = -(J^T J + r\Lambda^{-1})^{-1}(r\Lambda^{-1}\mathbf{p} - J^T\mathbf{v}) \ , \tag{2.7}$$

where r is a regularization term. The mean-shifts and the update are computed iteratively until convergence [21].

Despite the benefits of CLM-based methods, they have been outperformed recently. The reason why they have been outperformed is probably because of the local detectors that cannot model the complex variation of local landmark appearances, such as facial hair, expressions, and makeup. In a paper by Zadeh et al. [3], a new local detector called Convolutional Experts Network (CEN) was introduced. The CEN local detector is illustrated in Figure 2.3. CEN consists of a mixture of expert classifiers, each capturing different appearance prototypes without the need of explicit attribute labeling.

The Convolutional Experts Constrained Local Model (CE-CLM) combines the Constrained Local Model (CLM) with the Convolutional Experts Network (CEN) as a local detector [3]. The CE-CLM algorithm consists of two parts. In the first part, a response map is computed using the Convolutional Experts Network (CEN). The individual landmark alignments are estimated independently of the position of other landmarks. The landmark localization is done by evaluating the landmark alignment probability at individual pixel locations, see Equation 2.3. In the second part of the CE-CLM algorithm, the shape parameter is updated using a Point Distribution Model (PDM), see Equation 2.2.

When tracking the facial landmarks in videos, the CE-CLM is initialized based on the MTCNN landmark detection in the previous frame. Tracking drift is prevented with a CNN that reports if the tracking has failed based on currently detected landmarks. In the case of tracking failure, the CE-CLM is reinitialized with the MTCNN face detector [2].

The resulting 68 3D Facial Landmarks from the OpenFace 2.0 CE-CLM are illustrated in Figure 2.4. Two examples of the 3D Facial Landmark detection system in action on the test subject HPG487 specified in Table 3.2 are presented in Figure 2.5 and 2.6. Note that the landmark detection model does not lose tracking when the driver looks to the side.

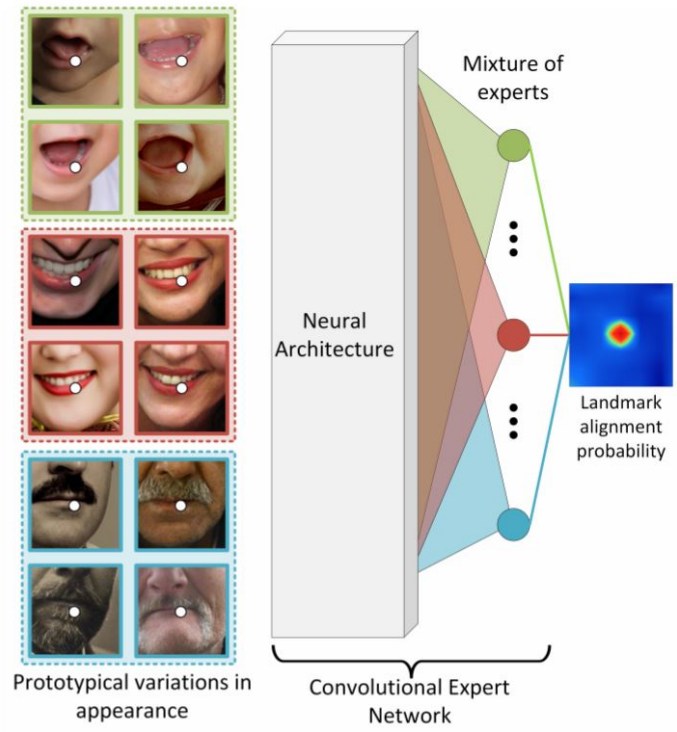

Figure 2.3: The appearance of a facial landmark naturally clusters around a set of appearance prototypes (such as facial hair, expressions, make-up etc.). In order to model such appearance variations effectively Zadeh et al. [3] introduce the Convolutional Experts Network (CEN). CEN combines the advantages of neural architectures and mixtures of experts to model landmark alignment probability. The image is taken from the article by Zadeh et al. [3].

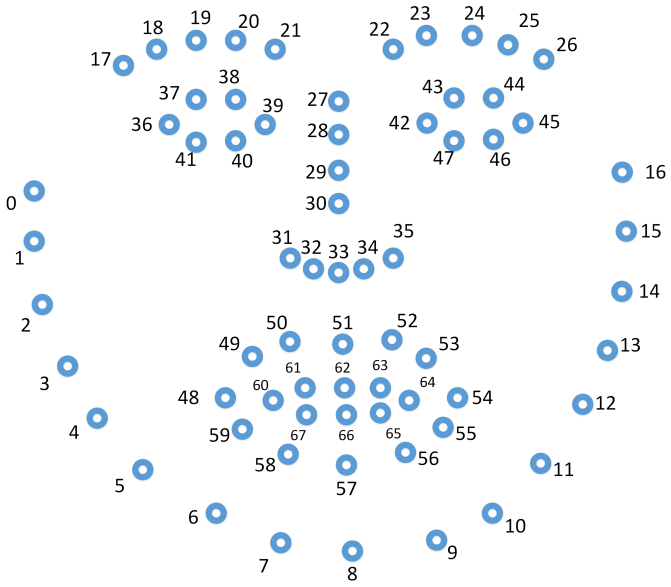

Figure 2.4: Facial landmark scheme from the OpenFace 2.0 CE-CLM. Source: [4].

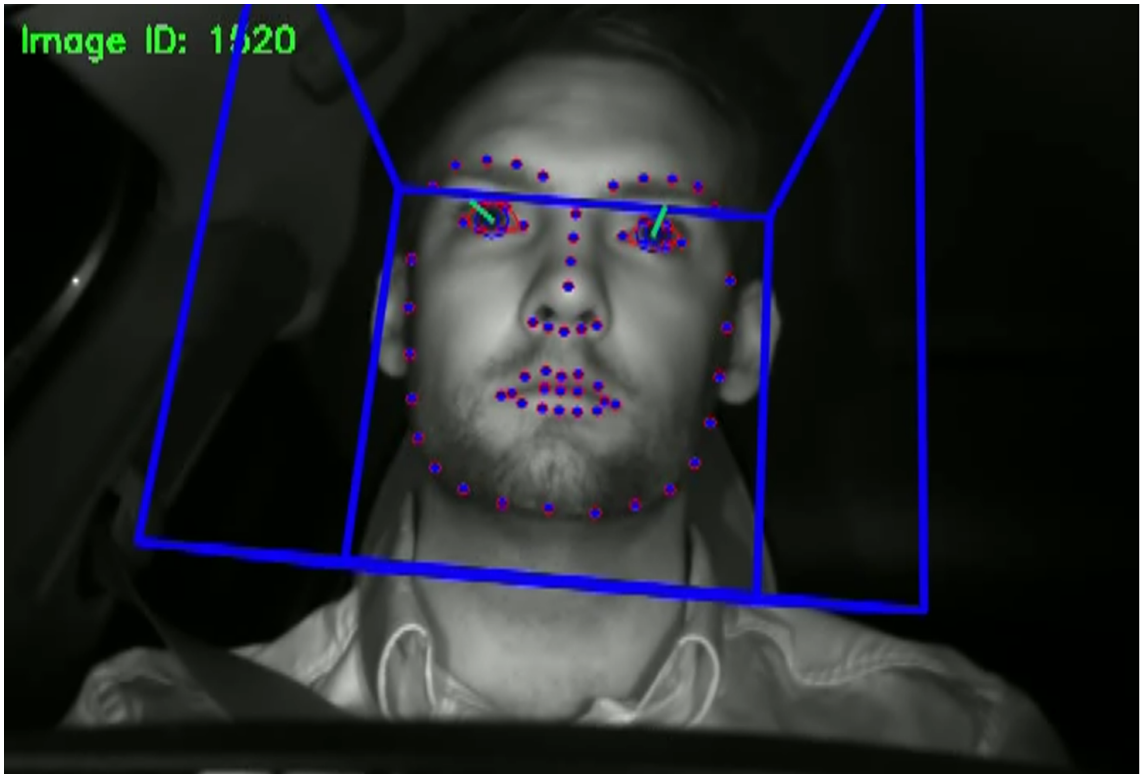

Figure 2.5: OpenFace 2.0 in action on driver HPG487.

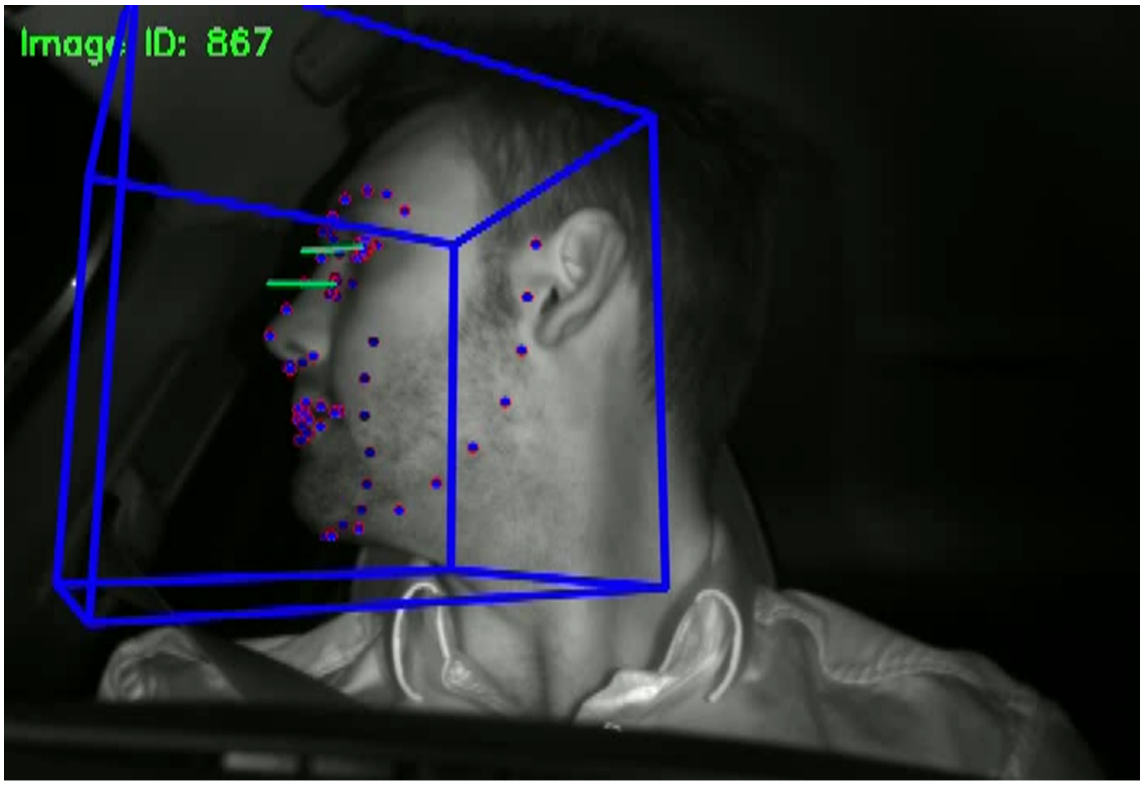

Figure 2.6: OpenFace 2.0 in action on driver HPG487.

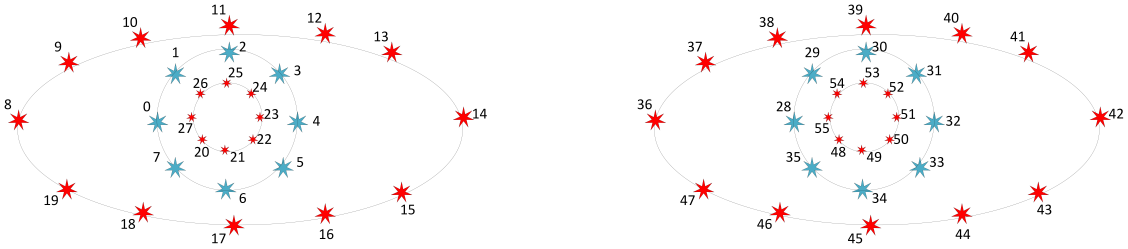

Figure 2.7: 3D eye region landmarks from the OpenFace 2.0 CLNF landmark detector. Source: [4].

2.3.3 Eye Gaze Estimation

OpenFace 2.0 eye gaze estimation is based on a Constrained Local Neural Field (CLNF) landmark detector able to detect eyelids, iris, and the pupil. The 3D eye region landmarks are illustrated in Figure 2.7 and can also be seen in action in Figure 2.5 and 2.6.

The CLNF landmark detector is an extension of the Constrained Local Models (CLMs) explained in Section 2.3.2 that uses more advanced landmark detectors and a more advanced fitting technique [22]. The more advanced landmark detectors are called Local Neural Field patch experts. These patch experts learn the non-linear and spatial relationships between pixel values and the probability of landmark alignment. Additionally, CLNF uses a Non-uniform Regularised Landmark Mean Shift, instead of the ordinary RLMS in Equation 2.5. This fitting technique also takes the patch reliabilities into consideration [21].

Based on the detected pupil and eye location landmarks, eye gaze vectors are computed for each eye individually [2]. A line is calculated from the camera origin through the center of the pupil in the image plane. By computing the line intersection with the eye-ball sphere, the pupil location in 3D camera coordinates is given. The estimated gaze vector is the vector from the 3D eye-ball center to the pupil location.

Based on the gaze direction vector, OpenFace calculates two gaze angles: x for horizontal gaze and y for vertical gaze [4]. The eye gaze angle vectors are the eye gaze directions in radians in world coordinates averaged for both eyes, with the camera being the origin. If a person is looking straight, both the x and y angles will be close to 0.

2.3.4 Facial Action Unit Recognition

The Facial Action Coding System (FACS) [23] is a commonly used method for coding facial behavior. It is an anatomically based system for measuring visually identifiable facial movements. The system defines 46 different Action Units (AUs). Although the FACS is anatomically based, there is not a 1:1 correspondence between AUs and muscle groups since a muscle may produce visually different actions depending on how it is contracted.

OpenFace 2.0 recognizes human facial movements by detecting FACS intensity and presence [2]. The action units that are recognized are presented in Table 2.4. According to [22] the performance of AU classifiers used in OpenFace 2.0 depends heavily on the training data and the ability to estimate facial expressions of a neutral face.

Table 2.4: Facial Action Unit numbers and the corresponding name in the Facial Action Coding System recognized by OpenFace 2.0. Carnegie Mellon University presents example images for each facial AU which can be found here: [5]. **For Lip Suck, the intensity value is not predicted by OpenFace 2.0 and hence cannot be used.

AU Number	**FACS Name**
1	Inner Brow Raiser
2	Outer Brow Raiser
4	Brow Lowerer
5	Upper Lid Raiser
6	Cheek Raiser
7	Lid Tightener
9	Nose Wrinkler
10	Upper Lip Raiser
12	Lip Corner Puller
14	Dimpler
15	Lip Corner Depressor
17	Chin Raiser
20	Lip stretcher
23	Lip Tightener
25	Lips part
26	Jaw Drop
28	Lip Suck**
45	Blink

OpenFace's AU detection system is based on the work of T. Baltrusaitis et al. [22], with slight modifications [2]. The system utilizes what is called appearance features and geometry features for detection of AUs. These features are person-normalized since the person's neutral facial expression must be known to determine the presence of AUs.

The appearance features are obtained by performing face alignment and calculating the Histogram of Oriented Gradients (HOG), after which Principal Component Analysis (PCA) dimensionality reduction is used. For the face alignment, the most stable facial landmark points are used (0 - 3, 13 - 16, 31 - 39, and 42 - 45 in Figure 2.4). These are similarity transformed to a representation of landmarks from a neutral facial expression, a mean shape from a 3D PDM. The similarity transformation is computed using Procrustes superimposition, which minimizes the Mean Squared Error (MSE) between aligned pixels. The resulting image is of 112×112 pixels where the distance between the centers of the two eyes is 45 pixels. Masking

is also done to remove non-facial information by computing a convex hull surrounding the aligned feature points. The HOG is then extracted from the 112 × 112 pixel image and dimensionality reduced with PCA keeping 95 % of the explained variability.

For the extraction of geometry features, a Constrained Local Neural Field (CLNF) model is used, which is already explained in further detail in Section 2.3.3. The geometry features are the landmark locations and the non-rigid shape parameters $\mathbf{\Phi}_i\mathbf{q}$ in Equation 2.2.

Next, to perform person-normalization of the appearance and geometry features, the person's neutral facial expression must be known. To do this, OpenFace 2.0 computes the median value of face descriptors in a video sequence of a person, which is then subtracted from the observed facial feature descriptors. Finally, AU presence and AU intensity are determined by using linear kernel SVM classification and SVR regression respectively. The use of linear kernels allows for real-time classification and regression.

2.4 Machine Learning

In this section, we first describe the theory behind the machine learning classifiers used in this project. Then, the theory behind training and evaluating the classification algorithms are covered.

2.4.1 Random Forest Classifier

A Random Forest classifier is a supervised machine learning model using ensemble learning. The Random Forest model fits several decision tree classifiers on the dataset. The three main components of a decision tree are internal nodes, edges/branches, and leaf nodes. Each internal node is labeled with an input feature. Its outgoing edges/branches lead to the possible values of the target or another internal node on a different input feature. The leaf nodes are terminal nodes that predict the target values, in the classification case, as class labels.

In the decision tree algorithm, one starts at the root node and split the data according to a criterion. In this project, the Gini impurity criterion is used. Gini impurity helps to quantitatively evaluate how good a split is by calculating the probability of incorrect classifications 2.8. The Gini impurity with C classes where $p(i)$ is the probability of classifying a datapoint as class i is calculated according to

$$G = \sum_{i=1}^{C} p(i) \cdot (1 - p(i)) \ . \tag{2.8}$$

As can be seen in Equation 2.8 above, a Gini impurity of $G = 0$ is the best possible value. $G = 0$ is achieved when the split is perfect. The best split is chosen in the decision tree algorithm by maximizing the Gini Gain. The Gini Gain is calculated

by subtracting the weighted impurities of the edges/branches from the original impurity [24]. When looking for the best split, a hyperparameter called max features can control the number of features to consider, for example, only looking at the square root of the total number of features. Another essential hyperparameter for the splitting process is the minimum number of samples required to split an internal node. Similarly, the minimum number of samples required to be at a leaf node can also be tuned as a hyperparameter [6].

In the Random Forest model, each decision tree gives a classification, and the result is averaged. The ensemble technique results in improved predictive accuracy and reduced variance [6].

If bootstrap is used in the Random Forest model, the decision tree classifiers only see a subset of the train dataset. Otherwise, the whole training dataset is used to build each tree. Another important hyperparameter is the number of estimators, the number of decision trees in the model. More decision trees lead to a more complex model, which might lead to overfitting. To prevent overfitting, one can limit the maximum depth of the decision tree [6].

2.4.2 Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN)

Artificial neural networks (ANNs) are structures of simple processing units called neurons [25]. The design is inspired by the structure of a real brain. Although there are many different types of neural networks, the fundamental properties remain the same. Every neuron in the network can receive multiple input signals, process them, and propagate an output signal to other neurons since each neuron is connected to at least one other neuron. The connection between two neurons has a weight, and in conjunction with an activation function, it determines the strength of any propagated signal. Thereby the weights reflect the importance of the different connections in the network. There is also a bias term which allows you to shift the activation function by adding a constant to the input signal. For supervised learning tasks, where the desired output is known, the back-propagation algorithm is used for training the neural network by iteratively adjusting the weight coefficients until the network output matches the target. The back-propagation algorithm utilizes the gradient descent optimization method to minimize the output error.

Adam optimization algorithm is an extension to regular stochastic gradient descent. It is well-suited for many different machine learning applications since it is computationally efficient for stochastic objective functions, problems with large datasets, and high-dimensional parameter spaces [26].

Recurrent neural networks (RNNs) are designed to learn sequential or time-varying patterns [27]. In RNNs, cyclic connections exist such that the signal from previously activated neurons are stored in the internal state of the network, which provides theoretically indefinite temporal contextual information [28]. This property differs from a regular Feed Forward Neural Network (FFNN), in which the signal can only propagate forward. However, standard RNNs fail to learn when the input data

consists of time lags greater than 5–10 discrete time steps between relevant input events and target signals [29]. This inability to learn is a consequence of the back-propagation through time (BPTT) technique used for training RNNs, which leads to vanishing and exploding gradients [28].

To solve this issue, the Long Short-Term Memory (LSTM) RNN was introduced, which includes special units called *memory blocks* [28]. These blocks in turn contain *memory cells* with self-connections that store the temporal state of the network, as well as *input-, output-*, *forget gates* which control the flow of neuron activations into and out of the cells.

`Pytorch` implements functionality for using a multi-layer LSTM RNN [30]. The implementation is based upon the works of Sak et al. [28] who defines a standard LSTM network that computes a mapping from an input sequence $x = (x_1, \ldots, x_T)$ to an output sequence $y = (y_1, \ldots, y_T)$ by calculating the network activations using the following equations iteratively from $t = 1$ to T:

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i) \tag{2.9}$$
$$f_t = \sigma(W_{fx}x_t + W_{mf}m_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.10}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{cx}x_t + W_{cm}m_{t-1} + b_c) \tag{2.11}$$
$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_{t-1} + b_o) \tag{2.12}$$
$$m_t = o_t \odot \tanh(c_t) \tag{2.13}$$
$$y_t = W_{ym}m_t + b_y \ , \tag{2.14}$$

where the W terms denote the weight matrices specifying neuron connections (e.g. W_{ix} specifies the weights from the input gate to the input), the b terms denote bias vector (eg. b_i is the bias vector for the input gate), σ is the logistic sigmoid activation function while i_t, f_t, o_t, c_t are the input gate, forget gate, output gate and cell state at time t respectively, m_t is the cell output activation vector and $\odot$ is the element-wise product. Often, m_t is referred to as the *hidden state* of the LSTM at time t.

Additionally, in `Pytorch`'s multi-layer LSTM implementation the input $x_t^{(l)}$ of the l-th layer (where $l >= 2$) is the hidden state $m_t^{(l-1)}$ of the previous layer multiplied with a dropout factor $\delta_t^{(l-1)}$, i.e. each $\delta_t^{(l-1)}$ is a Bernoulli random variable which is 0 with a specified probability [30]. Dropout is an adaptive regularization technique used to control overfitting by artificially corrupting the training data [31].

Although the recurrent architectures of LSTMs give the theoretical ability to retain information through sequences of indefinite length, it has been shown that this is not the case in practice. As an example, Bai et al. [32] found that an LSTM can fail to retain memory even for short sequences (< 100 time steps) when testing against a particular benchmark problem. The same report shows that Temporal Convolutional Network (TCN) architectures outperform LSTMs in many sequence modeling tasks and can maintain a much longer effective history than their recurrent counterparts.

2.4.3 Training and Evaluation

This section covers the general theory behind training and evaluation of machine learning classifiers and an introduction to a hyperparameter optimization framework used in the project.

2.4.3.1 Cross Validation

There are two primary steps for building a supervised machine learning model: training and evaluation. To circumvent overestimating the model performance, one must avoid using training data during evaluation. Instead, the available samples should be split into two clearly defined datasets for training and evaluation, respectively.

When the true distribution of samples in a dataset is unknown and the number of samples is limited, any possible split of the dataset into a training and evaluation set might not represent the underlying distribution. Therefore, when training and evaluating a model on such a split, the model's true performance might be greatly under- or overestimated, especially when the number of samples for the training and evaluation sets are small.

Cross Validation (CV) is a method for validating the performance of a machine learning model in a less biased fashion. The purpose is to encourage selecting a model that yields a trustworthy fit from the available samples [33]. In Leave-One-Out Cross Validation (LOOCV), one splits the data such that the model is trained on all available samples except one that is held out. The held-out sample is used as a single validation instance. Next, one performs a new split, where the following sample is held out for validation, and the previous held-out sample is included in the training data. By repeating this process until each sample has been validated once, the validation results can then be averaged to estimate the model performance.

In cases where computational resources are limited, and especially when the sample size is considerable, LOOCV can be infeasible to perform because of the long training times. Instead, it is common to use a *k-fold CV* approach in which multiple validation samples are grouped for each split. For example, 5-fold CV holds out 20% of samples for evaluation and 80% for training in each split, thereby resulting in 5 different trained models.

It is essential to realize that if the CV score is used for model selection, one needs to utilize a separate, withdrawn testing set for unbiased validation of the results. Otherwise, performance will be overestimated.

2.4.3.2 Early stopping for iterative optimization algorithms

Early stopping is commonly used in many iterative optimization algorithms to reduce the risk of overfitting [34]. Overfitting occurs when a machine learning model is trained to be highly accurate on the training data it is exposed to but is too sensitive to handle new and unseen data. With early stopping in neural networks, training is stopped once the performance measured on a held-out validation set stops increasing, rather than when the iterations are finished or when the performance measured on

the training set is optimal.

2.4.3.3 Metrics

In classification tasks, some important metrics are: accuracy, precision, recall, F1-score. These metrics are based on the concept of True Positives (TP), True Negatives (TN), False Negatives (FN), and False Positives (FP). Accuracy, recall, precision, and F1-Score can be defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} , \tag{2.15}$$

$$\text{Recall} = \frac{TP}{TP + FN} , \tag{2.16}$$

$$\text{Precision} = \frac{TP}{TP + FP} , \text{ and} \tag{2.17}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} . \tag{2.18}$$

In the case of multi-class classification accuracy with scikit-learn, the predicted class must match the corresponding ground truth class to be considered a correct classification [35]. For multi-class recall, precision, and F1-score with scikit-learn, an average weighted by the support (the number of true instances for each label) is a suitable option for imbalanced data [36, 37, 38].

It can also be insightful to study a confusion matrix. A confusion matrix C is constructed such that $C_{i,j}$ is equal to the number of observations known to be in the group i and predicted to be in the group j.

Another useful metric that is most commonly used for regression tasks is the Mean Squared Error (MSE) [39]. The MSE can be applied to some multi-class classification problems, like classifying the sleepiness level (such as the ORS scale) from 0 - 4, as

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(\text{ORS}^i_{\text{predicted}} - \text{ORS}^i_{\text{true}} \right)^2 . \tag{2.19}$$

The MSE metric considers how close the prediction was to the actual sleepiness value and penalizes large errors more.

When training neural networks for multi-class classification problems, the cross-entropy loss is commonly used as a loss function. It is useful since it probabilistically minimizes the classification error. The cross-entropy loss used in `Pytorch` [30] combines a log softmax and negative log-likelihood loss and is described by

$$\text{loss}(x, \text{class}) = -\log \left(\frac{\exp(x[\text{class}])}{\sum_j \exp(x[j])} \right) = -x[\text{class}] + \log \left(\sum_j \exp(x[j]) \right) . \tag{2.20}$$

2.4.3.4 Model optimization using Optuna

Most sophisticated machine learning models define several different hyperparameters, which require optimization to achieve desirable performance. The number of hyperparameters varies significantly from model to model. To reduce the number of tunable hyperparameters, the experimenter commonly chooses to fix some hyperparameters at a reasonable default. It is not unusual to have somewhere in the range of ten to fifty different hyperparameters for certain models [40].

In essence, hyperparameter optimization means searching for hyperparameter combinations resulting in optimal model performance. Hyperparameters can assume both discrete, categorical or continuous values depending on their corresponding usage in the underlying model. In many cases, the search space of hyperparameter combinations is enormous, which motivates a need for efficient algorithms with the capability of finding good combinations in a reasonable amount of time.

`Optuna` is an open-source hyperparameter optimization framework [7] which supports `Python`. The framework allows users to dynamically construct the parameter search space and implement efficient searching and pruning algorithms. See Listing 1 for a code example. More specifically, rather than conducting a random search for hyperparameter combinations, at each evaluation, one can instead sample a combination depending on the performance of previous combinations. If the evaluation seems unpromising, it can be pruned/terminated early on to free up resources for conducting another evaluation. In many other hyperparameter optimization frameworks, pruning is often overlooked. However, it is also an essential methodology under limited resource availability conditions. By optimizing the sampling and implementing efficient pruning strategies, the search time required for finding good hyperparameters combinations can therefore be decreased.

`Optuna`'s default sampler for choosing hyperparameter combinations in each new evaluation is based upon the Tree-Structured Parzen Estimator (TPE) approach, [40]. The approach conducts *independent sampling* which samples each parameter independently of each other, rather than *relational sampling*, which also considers and exploits correlations among the parameters. Although it might seem naive to assume that parameters have an independent relationship with each other, TPE is known to perform well even without using the parameter correlations [7]. The cost-effectiveness of the method should also be considered when computation resources are limited, making it a good choice in many cases.

In order to prune an unpromising evaluation early, an intermediate objective value can be reported to `Optuna` in every update step. Needless to say, pruning is only effective for iterative optimization algorithms for which an intermediate score can be computed. By default, pruning is determined according to the median stopping rule; prune the evaluation run if the best intermediate result is worse than the median of intermediate results of previous evaluations at the same step.

```
import optuna
import ...

def objective(trial):
    n_layers = trial.suggest_int('n_layers', 1, 4)

    layers = []
    for i in range(n_layers):
        layers.append(
            trial.suggest_int('n_units_l{}'.format(i),1, 128))

    clf = MLPClassifier(tuple(layers))

    mnist = fetch_mldata('MNIST original')
    x_train, x_test, y_train, ytest = train_test_split(
        mnist.data, mnist.target)

    clf.fit(x_train , y_train)

    return 1.0 - clf.score(x_test , y_test)

study = optuna.create_study()
study.optimize(objective , n_trials=100)
```

Listing 1: Example of Optuna's *define-by-run* style API, which allows users to construct the search space of hyperparameters dynamically. Here, the hyperparameters are the number of layers and the number of hidden units at each layer for a Multilayer Perceptron (MLP) classifier trained on the MNIST dataset. The example is taken from [7].

2.5 Data Management

To handle the large dataset, the `HDF5` (Hierarchical Data Format) high-performance data software library and file format can be used to manage, process, and store the images of the drivers. `HDF5` is built for fast I/O processing and storage [41]. Another convenient storage format is `JSON` (JavaScript Object Notation), which consists of key/value pairs. See an example in Listing 2.

3

Methods

This chapter describes the methodology from data collection to sleepiness prediction. The data collection section briefly explains how the dataset was collected by Volvo Cars. The data processing section explains how the drivers' dataset is processed into shorter video clips for training and evaluation. Next, the eye state classification process is covered. It describes the annotation process, as well as model and feature selection. Given the eye state classification model, it is possible to extract *eye blink features* such as the average blink duration. Sleepiness prediction from handcrafted eye blink features is the first approach to sleepiness prediction. Then, another sleepiness prediction approach based on temporal features of the face (such as gaze direction and micro-expressions) is described. Moreover, the methodology to evaluate the generalization ability of the sleepiness prediction models is covered.

3.1 Data Collection

Volvo Cars had acquired a dataset before the start of the project. The dataset consists of infrared (IR) video of drivers driving at night time. The IR illuminator on the SmartEye camera (facing the driver) has a wavelength of 850 nm [18] which lets the camera "see through" glasses and sunglasses of non-IR type [19].

In general, the drivers drive for two hours or until they fall asleep and are too tired to keep driving. The video data is collected in a real, naturalistic driving experiment and is manually annotated with sleepiness level logs of the driver. Data has been collected with ten different drivers, although only data from eight of them were possible to use for this project.

3.2 Data Preprocessing

The objective with the data preprocessing was to store all images in `HDF5` files [41], and all relating metadata in `JSON` files [42] to manipulate the data efficiently. Then, the goal was to create video clips (i.e., sequences of images) from this data to simulate short driving sequences.

3.2.1 Data Cleaning

Before creating the `HDF5` files containing the images and the `JSON` files containing the metadata, the dataset was processed to handle missing data, misaligned data and to trim the start- and end of the driving videos. This data cleaning was done using `MATLAB`. The following information was extracted for the drivers that had all the necessary data:

- `image_dir` - The directory the image is stored in
- `file_name` - The image file name of the frame
- `frame_number` - The number of the frame
- `frame_time` - The timestamp of the frame
- KSS - The KSS value of the frame
- ORS - The ORS value of the frame

The Swedish Institute of Computer Science (SICS) had pre-processed some of the data related to the Autoliv camera (side-facing) and the sleepiness logs containing the KSS and ORS value for each timestamp. Note that the sleepiness value for each timestamp is assumed to be equal to the next reported sleepiness level. The KSS and ORS updates for all drivers can be observed in Figure 3.1 and Figure 3.2. Note that some drivers do not experience all sleepiness levels during the driving session.

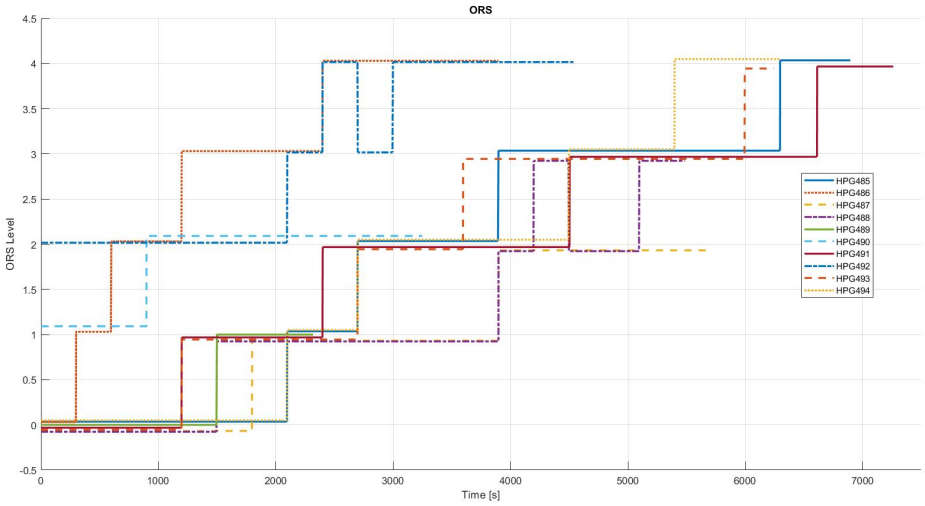

Figure 3.1: ORS level updates of all drivers during their driving sessions. Note that HPG486 and HPG490 were not included in the project.

However, since the SmartEye camera (facing the driver) was used in this report, some preprocessing steps were required to map the sleepiness annotations to the SmartEye images and their corresponding timestamps. First, the SmartEye images (`.PGM` or `.JPG`) were mapped to a file containing the frame number and timestamp

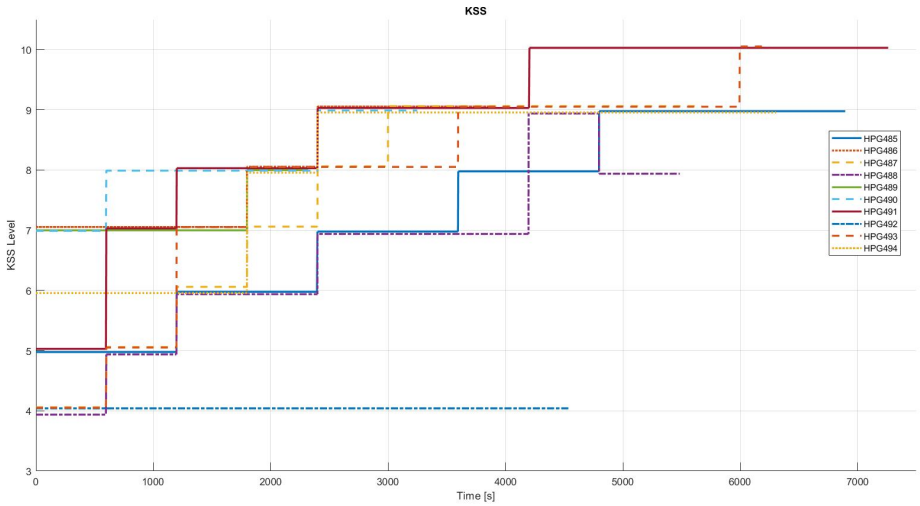

Figure 3.2: KSS level updates of all drivers during their driving sessions. Note that HPG486 and HPG490 were not included in the project.

for each image file. However, some image files did not have a corresponding frame number and timestamp, often a sequence of a few hundred consecutive images. The missing timestamps were imputed using linear interpolation [43]. Moreover, some drivers did not have any images at all. For these drivers, some additional data processing was required to extract the the SmartEye images (`.PGM` or `.JPG`) from files with `.sma` and `.smb` file extensions. However, two of the ten drivers had multiple `.sma` and `.smb` files and missing information about how to merge these. Hence, these two drivers were excluded from the dataset to avoid potentially incorrect sleepiness data. In total, this resulted in eight available drivers.

Another issue that arose with the timestamps is that they are relative to when the SmartEye camera started recording and not when the experiment started. Therefore, the SmartEye time axis was aligned with the Autoliv time axis since the Autoliv time axis already contained sleepiness annotations processed by SICS. To align the time axes, the images were converted to videos with their true frames per second (FPS) found by dividing the number of images by the total time duration. Matching points in the SmartEye and Autoliv videos were found by manual inspection. With the matching points, the SmartEye images and timestamps could be aligned correctly with the sleepiness logs (Observer Rated Sleepiness (ORS) and Karolinska Sleepiness Scale (KSS) values).

Next, the video was trimmed to remove undesired parts. The start trimming point was when the driver started driving, and the end trimming point was approximately 30 seconds after the last microsleep occurrence and close to the end of the driving session. These trimming points were determined by manual inspection of the videos. Finally, the images and timestamps were mapped to their corresponding sleepiness logs (ORS and KSS values).

3.2.2 Data Transformation and Storage

After processing the dataset in `MATLAB`, it was processed in `Python`. The images for a specific driver were stored in a `HDF5` file, each with a unique image id. For each image id, metadata is stored in a `JSON` file, see Listing 2. Note that the timestamps are given in seconds.

```
[{
    "image_id": 0,
    "internal_order": 0,
    "image_name": "20140403_000132_11149_1.PGM",
    "timestamp": 30.021362400000015,
    "driver_id": "HPG488",
    "kss": 4,
    "ors": 0
},
    ...
{
    "image_id": 396173,
    "internal_order": 396173,
    "image_name": "20140403_000132_414424_1.PGM",
    "timestamp": 6823.026557,
    "driver_id": "HPG488",
    "kss": 9,
    "ors": 4
}
]
```

Listing 2: Image metadata `JSON` example for driver HPG488.

From the image metadata, sequences of images were formed with a sequence duration of 30 seconds. Initially, these sequences contained the metadata described in Listing 3. The images were sampled at 30 FPS. However, the recording was recorded at close to 60 FPS for most drivers in the dataset. Sometimes, the frame rate in the camera dropped, and the time difference between two consecutive frames was larger than the sampling time. In this case, the previous image id was repeated. Sequences with more than 25 % duplicate images were removed. Moreover, sequences with images occurring four times or more in a row were removed.

```
[{
    "sequence_id": 0,
    "driver_sequence_id": 0
    "timestamps": [30.021362400000015, 30.05469573333335,
    30.08802906666668, ..., 59.95469573333335,
    59.988029066666684]
    "image_ids": [0, 1, 3, ..., 1427, 1429],
    "n_images": 900,
    "driver_id": "HPG488",
    "kss": 4,
    "ors": 0
},
    ...
{
    "sequence_id": 225,
    "driver_sequence_id": 225
    "timestamps": [6779.777616, 6779.810949333334,
    6779.844282666667, ..., 6809.710949333334,
    6809.744282666667]
    "image_ids": [393633, 393634, 393636, ..., 395387, 395389],
    "n_images": 900,
    "driver_id": "HPG488",
    "kss": 9,
    "ors": 4
}
]
```

Listing 3: Sequence metadata `JSON` example for driver HPG488.

3.3 Eye State Classification

One approach of classifying sleepiness was based on spatial blink features such as blink count and average blink duration. In order to extract these blink features, an eye state classification method was required to determine whether the eyes are open, partially closed, or closed in a particular frame. However, annotated image data was necessary to build an eye state classifier. This section describes the annotation process and the eye state classifier. The results are presented in Section 4.1.

3.3.1 Eye State Annotation

The definition of eye state described in Section 2.1.1.1 was used for the annotation process. An example of the different eye states for one of the drivers is presented in Figure 3.3.

Table 3.1 presents which drivers were annotated, the number of sequences, images

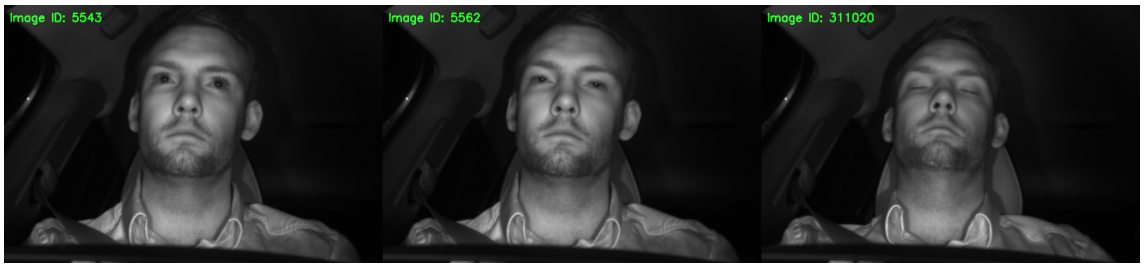

Figure 3.3: Examples of the three eye states for driver HPG487. Left: open eyes. Center: partially closed eyes. Right: closed eyes.

per driver, the distribution of open/partially closed/closed eye states. Since the sequences chosen for annotation were picked at different intervals for different drivers, this information is also provided. Moreover, Table 3.2 presents metadata about the annotated drivers in the dataset.

Table 3.1: Annotated data per driver including statistics about the eye state distribution for each driver.

Driver	Annotated Sequences	Total Sequences	Annotated Frames	Eye State Percentage 0/1/2	Annotated 30 seconds every X minutes
HPG485	15	144	13500	84.5/9.7/5.8	~ 5
HPG487	37	177	33300	94.9/1.7/ 3.4	~ 2.5
HPG488	23	226	20700	90.8/6.3/2.9	~ 5
HPG489	7	65	6300	89.7/7.5/2.8	~ 5
HPG491	12	234	10800	92.4/5.5/2.2	~ 10
HPG492	8	146	7200	78.9/13.4/7.7	~ 10
HPG493	11	201	9900	83.4/10.9/5.8	~ 10
HPG494	11	201	9900	81.1/7.8/11.0	~ 10
Total	124	1394	111600	89.1/6.3/4.6	-

The sequences of frames were converted to `.AVI` video files with the image id printed in the upper left corner. The videos were played at low speed using Windows Media Player. When blinks occurred, the playback speed was set to around one FPS to analyze each frame individually. The eye states were annotated for each image id in an Excel sheet for each sequence. To streamline this process, a `Python` script generated these Excel sheets with two columns: image id and eye state. A default eye state was set to open (0).

3.3.1.1 Inter-Annotator Agreement

The upper limit on a machine learning system performance is often the agreement between annotators. If annotators cannot agree with each other about the classification more than to some percentage, one cannot expect a computer to do any better [44]. Hence, the inter-annotator agreement was measured between the au-

Table 3.2: Metadata about the drivers in the dataset.

Driver	Gender	Glasses
HPG485	Male	No
HPG487	Male	No
HPG488	Male	Yes
HPG489	Male	No
HPG491	Male	Yes
HPG492	Male	No
HPG493	Male	Yes
HPG494	Female	No

thors of this report. Two drivers were selected as subjects for the study: HPG491 and HPG494. For both drivers, two 30-second driving sequences were annotated by both annotators. The results are presented in Section 4.1.1.

3.3.2 Classifying Eye State From a Moving Window of Frames

In order to classify the eye state for a current frame, it was expected that a classifier could utilize both the current and some previous frames. Three major design choices had to be addressed: the input features, the number of previous frames to consider, and the hyperparameters of the machine learning classifier. All these design choices were analyzed in an optimization study with `Optuna`. The most suitable model was selected from the study, considering performance and favoring simplicity rather than complexity. The results of this comparison are presented in Section 4.1.2.

3.3.2.1 Data Preparation

Given the annotated eye state data, a moving window of W (to be determined later) consecutive images is extracted with a step size of one image. Features extracted from the W images are used to predict the eye state of the last image in the moving window. In order to keep a fixed window size W, the first $W - 1$ images in a sequence are initialized with a closed eye state (0). See the part regarding eye state classification in Figure 3.7 in Section 3.4.2 for an illustration of this process.

Since the driver is not blinking most of the time, the distribution of the three different eye state targets is imbalanced, see Figure 3.4 for an example using $W = 6$. The class imbalance is handled by undersampling and oversampling according to Algorithm 1.

After balancing the distribution of eye states, the distribution of different moving window patterns was investigated. Using $W = 6$ again, in Figure 3.5, the eye state patterns for driver HPG487 for all manually annotated 30-second sequences are presented. Note that the most common pattern was [0, 0, 0, 0, 0, 0], which is a moving window of images where the driver has open eyes in all images. The second most common pattern was a window of images with closed eyes, [2, 2, 2, 2, 2, 2], which could be a long blink or a microsleep.

Algorithm 1: Balanced sampling of eye state classes.

Data:
$L0$: List of eyes open windows
$L1$: List of eyes partially closed windows
$L2$: List of eyes closed windows
begin
  $n1 \longleftarrow$ number of elements in $L1$;
  $n2 \longleftarrow$ number of elements in $L2$;
  if $n2 > n1$ **then**
    randomly draw $n2 - n1$ samples from $L1$ and add to $L1$;
  else if $n2 < n1$ **then**
    randomly draw $n1 - n2$ samples from $L2$ and add to $L2$;
  end
  assert that $n1 == n2$;
  $L0 \longleftarrow$ randomly draw $n1$ samples from $L0$;
  $n0 \longleftarrow$ number of elements in $L0$;
  assert that $n0 == n1 == n2$;
end

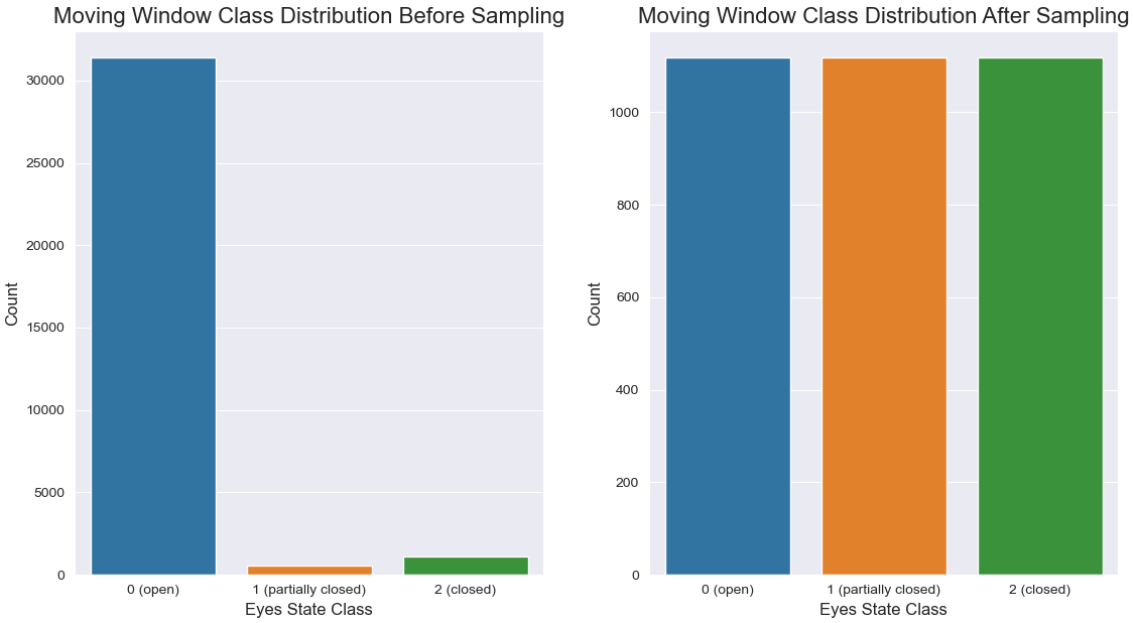

Figure 3.4: The distribution for a moving window of size $W = 6$ frames for the manually annotated 30-second sequences for driver HPG487. Note that the distribution between the three eye state classes is highly imbalanced since the driver mostly has his or her eyes open.

3.3.2.2 Choosing the Size of the Moving Window

The impact of W, the moving window size, was subject to analysis. The hypothesis was that if only one frame is used, the classifier might lose crucial historical information. However, using too many historical frames may introduce redundant

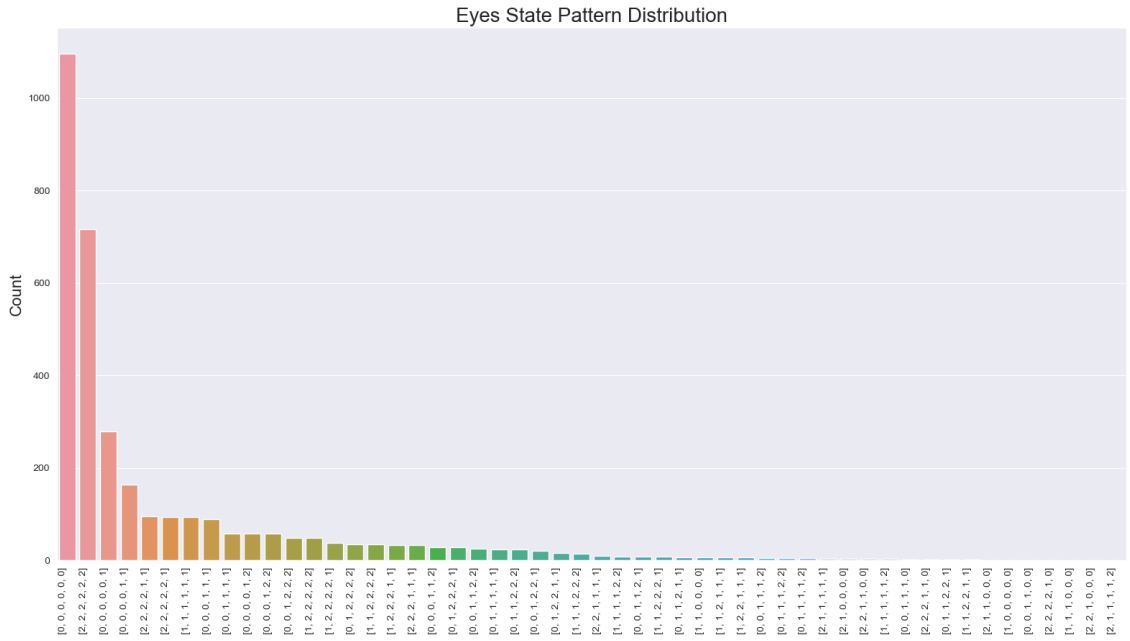

Figure 3.5: The eye state pattern distribution for a moving window of size $W = 6$ frames for the manually annotated 30-second sequences for driver HPG487. Note that the distribution between the different types of eye state patterns is highly imbalanced.

information. Therefore, different window sizes were evaluated, ranging from one up to ten frames.

3.3.2.3 Feature Selection

To predict whether the eye state is open, partially closed, or closed for a given frame the following four features were investigated based on OpenFace 2.0: *Eye Aspect Ratio (EAR)*, *blink regression intensity*, *gaze angle x* and *gaze angle y*. The Eye Aspect Ratio was calculated using the face landmarks around the eyes in Figure 2.4 with Equation 3.3 based on Equation 3.2 and 3.1 below:

$$EAR_{right} = \frac{\|LM_{37} - LM_{41}\| + \|LM_{38} - LM_{40}\|}{2 \cdot \|LM_{36} - LM_{39}\|} , \tag{3.1}$$

$$EAR_{left} = \frac{\|LM_{43} - LM_{47}\| + \|LM_{44} - LM_{46}\|}{2 \cdot \|LM_{42} - LM_{45}\|} , \tag{3.2}$$

$$EAR = \frac{EAR_{right} + EAR_{left}}{2} . \tag{3.3}$$

Note that LM_{37} = landmark no. 37. The blink regression value is the facial Action Unit (AU) from OpenFace 2.0 with AU number 45 defined in Section 2.3.4. The gaze angle x and gaze angle y from OpenFace 2.0 are defined in Section 2.3.3.

Table 3.3: Search space used during hyperparameter tuning of the random forest classifier. Note that the hyperparameter explanations are copied from [6].

Hyperparameter	Possible Values	Explanation
n_estimators	$\mathbb{Z} \in [1, 1000]$	The number of trees in the forest.
max_depth	$\mathbb{Z} \in [1, 100]$	The maximum depth of the tree.
min_samples_split	$\mathbb{Z} \in [2, 20]$	The minimum number of samples required to split an internal node.
min_samples_leaf	$\mathbb{Z} \in [1, 10]$	The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches.
max_features	auto, sqrt, or log2	The number of features to consider when looking for the best split.
bootstrap	True or False	Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

3.3.2.4 Choosing a Classifier

Initial studies indicated that tree-based classifiers seemed to work well for the eye state classification problem. Hence, a Random Forest Classifier from scikit-learn [6] was used and hyperparameter tuned. The parameter grid investigated are presented in Table 3.3.

3.3.2.5 Eye State Classification Optimization Study

An optuna parameter optimization study was performed. The parameters to determine were the moving window size W, the features, and the Random Forest Classifier hyperparameters. LOOCV was performed for the eight drivers to maximize the weighted average F1-score of the eye state classifications. Hence, the model was trained on seven drivers and tested on the remaining one in every split.

One of the best performing hyperparameter combinations, which also resulted in low model complexity, was then selected. Moreover, a LOOCV test with the selected hyperparameter setup was done using only 80 %, 60 %, 40 %, 20 %, 10 %, and 5 % of the available data. This data reduction test gave some clues about how much data is necessary to build an eye state classification model. Since the data was imbalanced between the different drivers, X % of each of the drivers' data was sampled randomly to maintain the same proportions.

With the selected hyperparameters, a model was fitted with all available data. Since the model was trained on all available data, it was trained on different amounts of

data from each driver, which should be considered when evaluating any results. Moreover, this model had an unknown performance since there was no validation or test set. Therefore, an ordinary train-test split was done with the test size being 1/8:th of the data to get around this issue. In this case, the same drivers were used in both the training and test set. Therefore, the train-test split on all drivers was more of a sanity check to see that the performance would not drop when adding the final driver to the data.

Additionally, the hyperparameters found during the optimization study were used to train one classifier for each driver. Each model was trained on 70 % of the driver's data and tested on 30 % of the data. The performance was compared to the performance of the LOOCV model to get a sense of the generalization ability.

The chosen eye state classifier was then used to predict the eye state of the sequences that were not manually annotated in an unsupervised fashion. The unsupervised annotation led to several incorrect annotations. However, manually annotating all video sequences would take too much time.

3.4 Sleepiness Prediction

Taking a step back, recall that the main objective was to classify a driver's ORS or KSS level given an image sequence of a certain length. Each blink feature as defined in Section 2.1.1.2 was calculated in relation to this image sequence. For example, *PERCLOS* is the percentage of the sequence for which the driver has a closed eye state. In contrast, *average eyes opening duration* refers to the average time it takes the driver to open his or her eyes, calculated over all the occurring blinks in the sequence.

Notably, using such blink features for classifying the driver's sleepiness level means that temporal time-series features are squeezed into single spatial values. In essence, the time series of different features computed from the video sequences are condensed into a few spatial data points for a video sequence of a specific length. We refer to this method of using the extracted blink features for sleepiness prediction as the *sleepiness from handcrafted eye blink features approach*, and it is further explained in Section 3.4.2.

Similar approaches seem to be the norm for classifying sleepiness from video data. However, it does not seem unlikely that another approach could be to predict sleepiness directly from the time-series data, rather than heuristically choosing spatial features beforehand. RNN-based models, such as the LSTM, are designed to handle sequential data. Thus, it might be possible for such models to learn to classify sleepiness on the raw time series. We call this method of performing predictions directly on time-series data the *sleepiness from temporal features approach*, and it is explained further in Section 3.4.3. Both approaches have a lot in common, which is why this section first covers a general description of the experimental setup.

Except for the two sleepiness prediction models, two baseline predictors were implemented: a constant predictor and a random (uniform) predictor. Reporting the baseline of a constant predictor tells us how much the system can learn. The baseline predictors were generated using 120-second sequence durations. The sleepiness prediction results are presented in Section4.2.

3.4.1 Experimental Setup

As explained in Section 2.1, even with clear and distinct sleepiness definitions, it is difficult to determine the real sleepiness level of a person both for an external observer and for the subject whose sleepiness is being evaluated. Therefore, one can suspect that the data's sleepiness labels might be slightly incorrect. Additional reasoning strengthening this assumption is further explained in Section 3.4.1.3. If this is the case, it would be hard, if not impossible, for any machine learning model to predict the exact target label correctly every time. Thus, having many different targets for sleepiness prediction is reasonably not any better. One can expect the error rate of the underlying annotated data to increase with the resolution of the sleepiness scale used.

In addition to this fact, performing multi-objective optimization (both ORS and

KSS) is harder and more time-consuming, and there was no justification found as to why such an approach would result in a fairer estimation of the drivers level of sleepiness. Therefore, it was decided to perform sleepiness prediction towards the ORS scale only, thereby disregarding the KSS labels entirely. Another reason for choosing ORS over KSS, was that the update frequency of ORS was double that of KSS.

Furthermore, in addition to choosing an optimization target, there are also many different possible metrics one can choose to optimize, as explained in Section 2.4.3.3. We choose to evaluate models based on the Mean Squared Error (MSE), where high performance means a low MSE. This metric has interpretable results in this case, as one gets to see how far off the model is when classifying the sleepiness targets. For example, with an MSE of 1.0, a model classifies within a ± 1 target accuracy, which should be considered in regards to the five different ORS targets.

Next, the experiment had to be designed such that an unbiased estimation of the system performance could be calculated. The data was limited to eight drivers. Therefore, the only reasonable way to obtain an unbiased estimate was to utilize Leave-One-Out Cross Validation (LOOCV) by validating on one held-out driver and training on the rest in every split. By averaging the MSE results from each split, the estimated performance of classifying sleepiness on a completely new and unseen driver can be calculated. However, model selection and hyperparameter tuning are also crucial to ensure one obtains a working model. The model selection and hyperparameter tuning also need to utilize LOOCV, due to the across-driver variance in appearances, sleepiness behavior, and other such varying patterns.

Crucially, choosing the best model/hyperparameter setup with respect to the corresponding LOOCV score means that a bias is introduced. The performance would be overestimated in regards to the general population. The overestimation happens because the data used for model evaluation (the held-out testing drivers) is also used for model selection by following this approach. A simple and straightforward approach for obtaining an unbiased estimation of model performance could be used if more data were available. For example, if 100 drivers were available, one could randomly select 70 drivers for model selection. The found model's performance could be evaluated on the remaining 30 drivers, which probably would approximate the underlying population quite well. Instead, when only eight drivers are available, one driver needs to be held out entirely from the model selection and hyperparameter optimization process using *nested LOOCV*.

Following the nested LOOCV approach means that for each outer LOOCV split, seven drivers are used for model selection/hyperparameter optimization (by carrying out an inner LOOCV procedure on the seven drivers), and the remaining driver is used for testing. As the inner procedure never gets to access the test driver, the evaluation would be unbiased for that particular driver. However, any individual test driver does likely not approximate the general population's distribution. Therefore, a significant variance would be introduced in the results.

Note that the outer LOOCV loop yields eight different models. They will each have

a significant variance present in the test result, although the result is unbiased. In order to obtain a fair estimation, one could imagine constructing an ensemble classifier. For example, this classifier could employ majority voting to decide a driver's level of sleepiness among the eight models. An approximate but fair estimate of the performance of the ensemble classifier can be obtained by averaging the performances of the eight individual models.

However, in addition to estimating the performance of such an ensemble classifier, we also wished to obtain the best single Random Forest classifier we could find. A single Random Forest would be less complex and easier to work with when it comes to estimating the effect of adding or removing drivers from the dataset.

3.4.1.1 Implementation of Experimental Setup

Our experimental approach first utilized regular nested LOOCV to find a set of promising hyperparameter setups. Then, a validation score of the system performance for each setup was calculated through what we define as Method A. Next, Method B was used for finding an unbiased test performance of the system for the same set of hyperparameters. The results of Method B could also be used to estimate what the performance of an ensemble classifier would be for a general driver.

These methods are summarized by Table 3.4 and Figure 3.6. In the table, each driver is coded by the numbers $[1, \ldots, 8]$. Eight different hyperparameter setups, which were obtained by nested LOOCV, are denoted by $[\mathrm{S1}, \ldots, \mathrm{S8}]$. These setups were obtained by running an `Optuna` hyperparameter optimization study of 250 trials (1 trial = 1 evaluated hyperparameter combination) for each outer LOOCV split. Each trial consisted of an inner LOOCV process, for which an average MSE was calculated. Picking the best trials for the respective studies resulted in the hyperparameter setups $[\mathrm{S1}, \ldots, \mathrm{S8}]$.

Note that this procedure in total required $8 \times 250 = 2000$ inner LOOCV processes on 7 drivers, resulting in 14000 different model fits. With large hyperparameter search spaces, the problem is quite intractable for models that generally require a long training process. A simpler way to think about it is that for each additional hyperparameter combination that we wish to evaluate, $7 \times 8 = 56$ more fits are required. That is because the number of trials for each respective outer fold should be the same.

Table 3.4: Experiment setup for obtaining a model with appropriate hyperparameters and unbiased performance estimations.

Driver→ Setup↓	**1** (HPG485)	**2** (HPG487)	**3** (HPG488)	**4** (HPG489)	**5** (HPG491)	**6** (HPG492)	**7** (HPG493)	**8** (HPG494)
S1	Test	Train	Train	Train	Train	Train	Train	Train
S2	Train	Test	Train	Train	Train	Train	Train	Train
S3	Train	Train	Test	Train	Train	Train	Train	Train
S4	Train	Train	Train	Test	Train	Train	Train	Train
S5	Train	Train	Train	Train	Test	Train	Train	Train
S6	Train	Train	Train	Train	Train	Test	Train	Train
S7	Train	Train	Train	Train	Train	Train	Test	Train
S8	Train	Train	Train	Train	Train	Train	Train	Test

Method A: Going into this method, $[S1, \ldots, S8]$ were known to be promising hyperparameters. However, the estimated performance was subject to some randomness introduced when balancing training data. Thus, we re-ran the 8 trials corresponding to the selected hyperparameter setups 10 times to obtain an average LOOCV MSE. As an example, let us walk through the calculation of the LOOCV MSE for setup S1, shown in Figure 3.6.
First, examine the row spanned by S1 and the 1st outer loop fold. Here, driver 1 is considered the test driver and is excluded entirely. A LOOCV procedure on the remaining drivers $[2, \ldots, 8]$ is carried out 10 times, using parameter setup S1. Therefore, the resulting average MSE is calculated on training drivers only. We refer to it as the *setup loss* of parameter setup S1 to differentiate it from scores obtained by evaluating on test drivers. This process is then repeated to calculate the setup loss of S2. For S2, the training drivers are $1 \cup [3, \ldots, 8]$ and so on.

Method B: To perform a test on the held-out testing driver from the outer loop, a final fit would be required. We choose to make this fit over all of the 7 training drivers for each respective outer LOOCV fold. Again, let us consider S1 as an example and walk through the calculation of its unbiased testing score. With S1, a model is fitted on the first fold containing all drivers except driver 1. Then, the MSE is calculated by predicting on driver 1. Just as in Method A, this fit and evaluation are performed ten times to reduce the impact of the random sampling used to balance the training data. Thus, an average testing MSE of S1 on driver 1 is obtained.
Since driver 1 was never used to optimize the hyperparameters or fit the model, this score is completely unbiased. Continuing with the same pattern, this approach results in eight different unbiased testing scores. However, the scores are all obtained by testing on single drivers, and thus a large variance is included. Recall that one could consider the scores as if individual classifiers of an ensemble model obtained them. Then, one could calculate the average to obtain an approximate system performance for an ensemble classifier that is not skewed towards any particular driver.

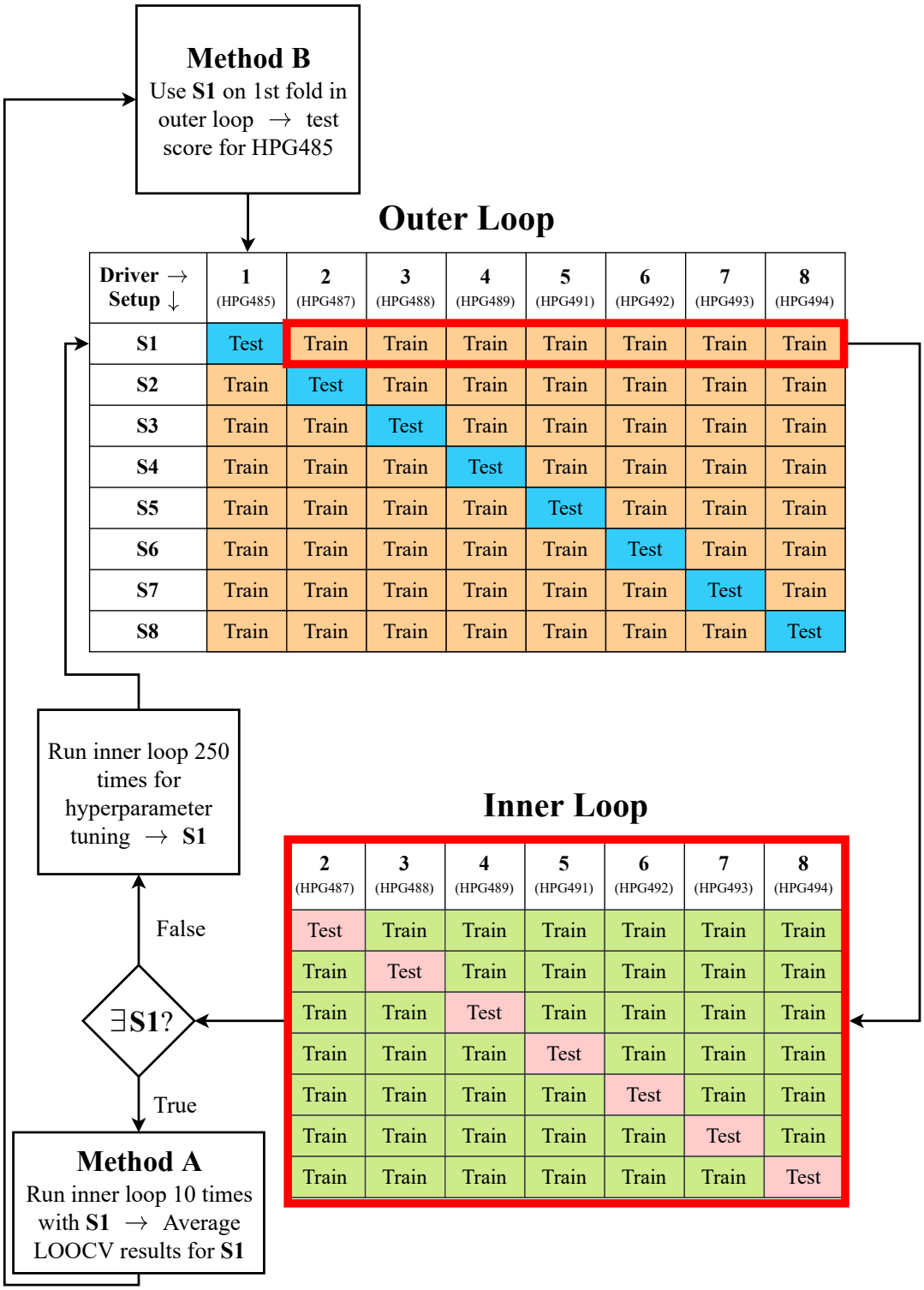

Figure 3.6: Nested leave-one-out cross-validation experiment for obtaining setups [S1, . . . , S8] and the resulting performance estimations from Method A and Method B. The figure only illustrates how S1 is obtained and its performance estimated with Method A and Method B. However, the process is similar for the remaining setups [S2, . . . , S8], with the difference being the outer loop split.

Next, the effect of adding or removing drivers from the training data was of interest to evaluate. For example, what might happen to performance if training occurs on five drivers instead of seven, or maybe even when training on just one driver? To investigate this effect, we opted to select a single hyperparameter setup out of $[S1, \ldots, S8]$ rather than using the method of ensemble classifiers. This choice was made in order to reduce the complexity of the experiment and the number of necessary fits. Thus, we define Method C, which explains how a particular setup was selected from $[S1, \ldots, S8]$, which was later used in the generalization experiment explained in Section 3.4.1.2.

Method C: As previously mentioned, Method B yielded unbiased estimations of the performance for each setup, but with a high degree of variance since the test results are influenced by the current testing driver. In order to compare $[S1, \ldots, S8]$ against each other, they had to be evaluated on more than one driver to reduce the variance. Therefore, it is clear that we had to base the selection of setup according to the setup loss obtained in Method A.
Again, let us consider an example to explain this method. Assume the best scoring setup according to the setup loss is S1. The problem remains that S1 has only been evaluated on driver 1. Thus, there is a large degree of variance. However, one can imagine a simple solution; use S1 and fit a model according to the data split of the other setups, then calculate and average the scores over the respective test drivers. For example, using S1 on S2's data split, we fit a model with S1 on drivers $1 \cup [3, \ldots, 8]$ while testing on driver 2. The variance is reduced by also doing the same for the other data splits and averaging the results.
However, it is important to realize that a slight bias is then introduced. The performance while testing S1 on S2's dataset split is slightly overestimated since the hyperparameter optimization process for selecting S1 included driver 2. Therefore, the hyperparameters of S1 work better than expected when testing a model with S1's parameters on driver 2, even though that model is only shown the other drivers.
Nonetheless, we assume the benefits of reducing the variance outweigh the downsides of a slightly increased bias, but it is necessary to recall this assumption when evaluating the results. Note that the results here were also averaged by running the Method ten times to reduce the impact of balanced random sampling applied to the training data in each split.

3.4.1.2 Effect of Adding Additional Training Data

One can compare how a sleepiness prediction model and parameter setup are affected by adding or removing one or more drivers from the training data. We call this the generalization ability of the model. The generalization ability could give an indication (a biased estimation) of how many drivers would be needed for a generalized model capable of accurately classifying sleepiness on the average person.

In Method C, the average test MSE per driver is obtained by calculating the MSE in each LOOCV split. Notice that this estimates the hyperparameter setup's average

Table 3.5: Cross validation split combinations used to evaluate the generalization ability of the sleepiness prediction model.

Name of Test	Test set size (No. of drivers)	Train set size (No. of drivers)	No. of Combinations
LOOCV	1	7	8
L2OCV	2	6	28
L3OCV	3	5	56
L4OCV	4	4	70
L5OCV	5	3	56
L6OCV	6	2	28
L7OCV	7	1	8
Total	**-**	**-**	**254**

performance when training with seven drivers and testing with the remaining one. Instead, one could also imagine obtaining an estimate of the average performance when training with six drivers and testing with two and so on. The complete opposite case is also possible; training with one driver and testing with seven. Note that this procedure is very similar to Method C, where the only differing property is the changing number of test drivers.

Therefore, we carried out Leave-X-Out Cross-Validation (LXOCV), meaning that X drivers were used for testing and the remaining ones for training in each split. Then, the impact of decreasing the number of training drivers could be evaluated.

When using one test driver, there are only eight possible combinations one could pick, but as soon as more drivers are used for testing, the number of possible combinations grows. Table 3.5 lists the number of combinations given the number of test drivers. As an example, if there are three drivers in the test set, there are $\binom{8}{3} = \frac{8!}{(8-3)!\cdot 3!} = \frac{8\cdot 7\cdot 6}{6} = 56$ train/test split combinations for L3OCV.

In total, there are 254 possible combinations. For each combination, the best hyperparameter setup (according to the setup loss obtained in Method A) was fitted to the training data, and the MSE was calculated from the predictions on the test data. This procedure was repeated ten times just as in Method C to reduce the impact of the random sampling used to balance the training data, and an average MSE was calculated for each combination. The results indicate the performance difference for varying the number of training drivers. However, note that the testing sets are slightly different when varying the number of held-out drivers, meaning that the results should be interpreted with caution.

3.4.1.3 Finding a Suitable Sequence Duration

It was assumed that the sleepiness prediction performance would rely on the chosen sequence duration. Earlier works in driver sleepiness prediction use different sequence durations, ranging from a few frames up to ten minutes. Nevertheless, some fundamental limitations on the sequence durations to be evaluated can be determined. First, the ORS level was noted every five minutes during data collection.

Hence, it would not make sense to use longer sequences. Using longer sequences could lead to overlapping sleepiness levels within sequences when the ORS level change.

Moreover, the extracted features from the *handcrafted eye blink features approach* (explained later in Section 3.4.2) have a high variability for short sequences. For example, consider a five-minute sequence when the driver is very sleepy. In the first four minutes, the driver struggles to keep awake, but the microsleeps only occur during the last minute. If we divide this sequence into one-minute segments, each corresponding to the same ORS level, any feature such as microsleep_count or PERCLOS would be low for the first four segments. In contrast, the fifth segment contains extremely sleepy behavior instances. Therefore, increasing the sequence duration makes these types of features more robust.

Another potential hazard of using sequence durations much shorter than the ORS update interval (five minutes) is that the accuracy of the annotated labels is reduced. The label quality is likely to decrease with a decreasing sequence duration, which leads to outliers and incorrectly annotated data points. With this in mind, it was decided that sequence durations of 30, 60, 120, 180, and 240 seconds were appropriate values to be evaluated. Thereby, the sequence duration could be treated as a tunable hyperparameter. Further reasons for choosing these sequence durations are given in Section 3.4.1.4.

3.4.1.4 Data augmentation

As previously mentioned, each driving experiment was conducted for a maximum of two hours per driver, and the ORS level was annotated every five minutes during the data collection process. Therefore, assuming that the two hours are split into segments at each five-minute interval, one would obtain a total of only 24 observations per driver. Having 24 observations per driver is far from enough training data for many different models, considering only eight drivers are available.

However, it is possible to augment the data by shifting a moving window over the driving sequence. Each new shift is unique, as at least one new frame is added and one old frame is removed in every shift. The number of frames added and removed depends on the step size of the shifting window. Shifting a moving window is similar to data that would be available during a real-world scenario, in which a camera would capture a new image with a specific frequency.

In order to augment the data by shifting, the moving window size needed to be shorter than five minutes since otherwise, targets could be overlapping whenever the ORS level would change. That is also why in Section 3.4.1.3, the chosen sequence durations were specified to be 30, 60, 120, 180 and 240 seconds. Notice that these sequences are shorter than five minutes (300 seconds). Having sequences shorter than five minutes meant that the moving window for all the different sequence durations could be shifted.

Moreover, the larger the moving window size, the fewer shifts are possible if the step size is held constant. For example, using a step size of one second for a moving

window of 240 seconds, only 60 shifts can be performed until the five-minute mark is reached. If a moving window of 30 seconds is used, 270 shifts can be performed instead.

Furthermore, an equal amount of augmented samples for all moving window sizes is favorable to obtain a less biased estimation (towards the amount of data) of the optimal sequence duration. To guarantee an equal amount of samples, one must reduce the step size proportionally to increased window size. However, this comes with a trade-off; as step size decreases for each increasing window size, the duplicated data in each step also increases. As an example, with a step size of one frame, there will only be a one-frame difference between two sequential steps of the moving window. These two observations would be almost identical and would not provide much additional value to the training process. On the other hand, if the step size would be equal to half the length of the moving window, each new step would contain 50 % previously seen frames and 50 % new ones, although then not as many steps are possible to take. The moving window step size / shifts are presented in Table 3.6.

Table 3.6: Data augmentation details with different sequence durations and moving window step sizes. There are always 31 training instances and 12 test instances for each data augmentation combination of sequence duration and moving window step size.

Sequence Length [seconds]	**Moving Window Step Size [seconds]**	**Percentage Data Repeated [%]**	**Training Instances (31)**	**Test Instances (12)**
240	2	99.17	00:00 - 04:00 00:02 - 04:02 ... 00:58 - 04:58 01:00 - 05:00	00:05 - 04:05 00:10 - 04:10 ... 00:55 - 04:55 01:00 - 05:00
180	4	97.77	00:00 - 03:00 00:04 - 03:04 ... 01:56 - 04:56 02:00 - 05:00	01:05 - 04:05 01:10 - 04:10 ... 01:55 - 04:55 02:00 - 05:00
120	6	95.00	00:00 - 02:00 00:06 - 02:06 ... 02:54 - 04:54 03:00 - 05:00	02:05 - 04:05 02:10 - 04:10 ... 02:55 - 04:55 03:00 - 05:00
60	8	86.66	00:00 - 01:00 00:08 - 01:38 ... 03:52 - 04:52 04:00 - 05:00	03:05 - 04:05 03:10 - 04:10 ... 03:55 - 04:55 04:00 - 05:00
30	9	70	00:00 - 00:30 00:09 - 00:39 ... 04:21 - 04:51 04:30 - 05:00	03:35 - 04:05 03:40 - 04:10 ... 04:25 - 04:55 04:30 - 05:00

3.4.2 Sleepiness From Handcrafted Eye Blink Features

As explained in Section 2.1.1, common indicators of sleepiness such as a high PERCLOS and similar features based upon movements of the eyelids have previously been utilized to detect driver drowsiness. In order to assess such an approach for the given dataset, the best eye state classifier obtained from the study described in Section 3.3.2.5 was used to classify whether the driver's eyes were open, closed, or partially closed for all images in each sequence.

Given the eye state of each sequence and each driver, the 36 different eyes blink features introduced in Section 2.1.1.2 were calculated and standardized by removing the mean and scaling to unit variance. Note that the two initializing zeros for the window size $W = 3$ were included in the eye state sequence when calculating the blink features for the sequence. A better approach would be to drop these zeros and only work with fully visible sequences. However, the consequences of this choice have a minimal impact on performance, as it leads to $\sim$0.22% of the eye state values for each frame obtaining a $\sim$10% risk of being classified wrong on average (since for a 30-second sequence of 30 FPS, the initial 2 out of 900 total frames are assumed to be open, and the eyes are fully open in $\sim$90% of the images according to our annotated material).

The calculated eye blink features were then used as input to a sleepiness classifier. An overview of the complete sleepiness prediction pipeline using this approach is presented in Figure 3.7.

Note that the eye state classifier was trained on all available annotated data from all drivers. Thus, when classifying the sleepiness on a driver, the eye state model had already been trained on that driver. Hence, one could argue that any sleepiness prediction results obtained might be slightly overestimated relative to what could be achieved for a completely new driver. However, the eye state classifier and the sleepiness classifier are in reality unrelated to each other, which means the actual performance of the eye state classifier is the only parameter affecting the sleepiness prediction performance. Towards the goal of classifying sleepiness, it does not matter what data was used for training the eye state model as long as the extracted eye blink features contain sufficient sleepiness information. The results of this approach are presented in Section 4.2.1.

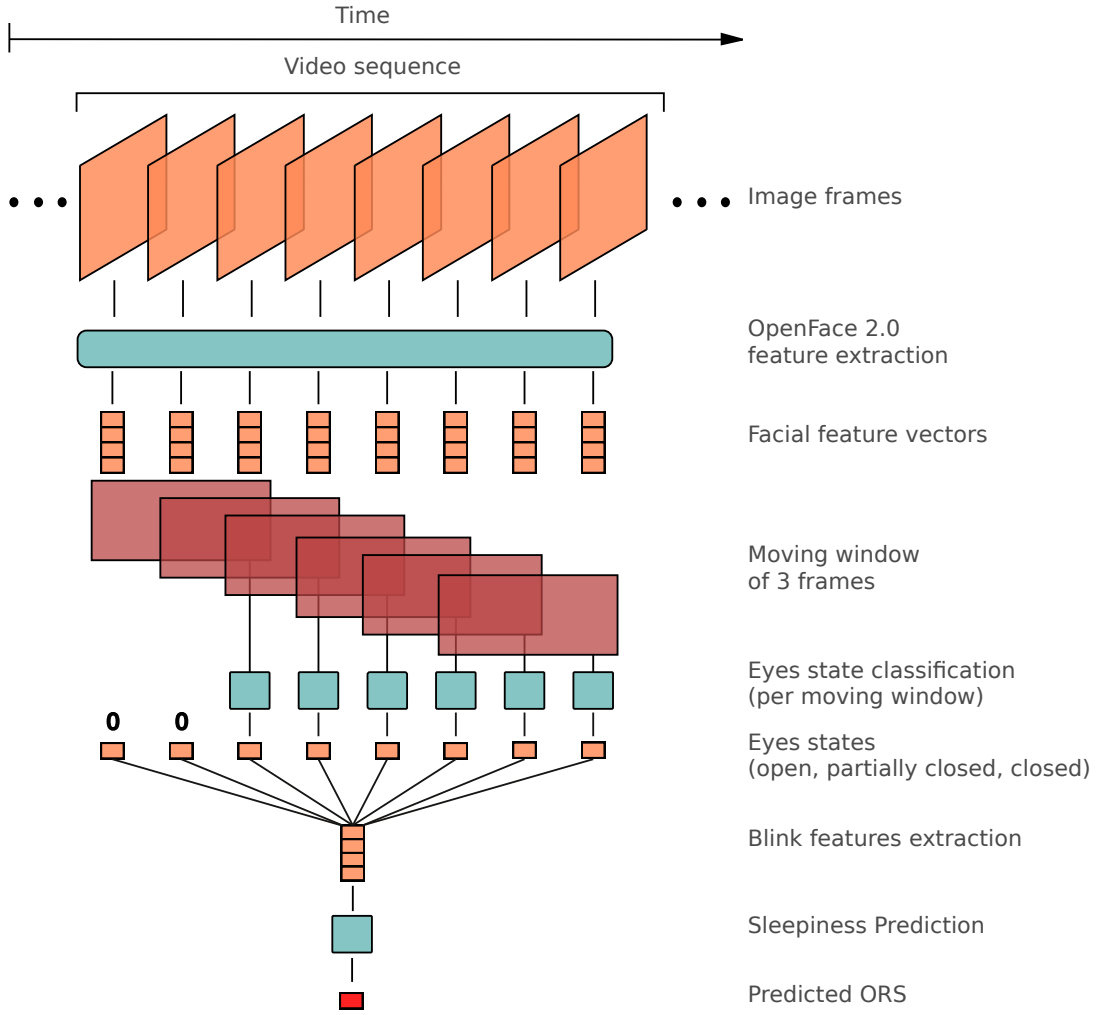

Figure 3.7: High-level illustration depicting how the temporal image sequences are first processed by OpenFace 2.0, after which the eye states in each frame is detected as explained in Section 3.3.2. Then, blink features from Section 2.1.1.2 are used to classify sleepiness.

The next task was to find a suitable classifier capable of determining driver sleepiness based on the extracted eye blink features. Generally, initial results indicated that tree-based methods performed quite well for the task at hand. Specifically, it was decided to proceed with a Random Forest as a sleepiness classifier. Note that the initial results that motivated this choice were based on biased performance scores. It would be too time-consuming to evaluate many different models as rigorously as required to make an unbiased model selection choice. Additionally, there were a few other reasons as to why a Random Forest classifier was deemed as an appropriate model in this case:

I. Out of the 36 handcrafted eye blink features, only a few are likely important predictors of sleepiness. Random Forest models have built-in feature selection; there would therefore be no need to perform a study for picking out a handful of features as the performance would not be highly affected for this type of model.

II. Random Forests are easy and fast to train. Given a large number of required fits to evaluate the model as specified in the experimental setup (see Section 3.4.1), it would be impossible to test a model that requires a very time-consuming training process.

III. With a Random Forests, it is easy to adjust the bias-variance trade-off. A large depth and many estimators yield a highly complex model, which can reduce bias but increase variance. With a small depth and a few base estimators, variance can be reduced at the cost of increasing bias.

After choosing the classifier type to be a Random Forest, hyperparameter tuning and performance estimations were obtained by following the experiment setup outlined in Section 3.4.1. After conducting the experiments of Method A, Method B and Method C, a final experiment Method was defined for exploring the feature importances. We refer to this as Method D.

Method D: This method did not utilize nested cross-validation but instead used regular LOOCV in which each split consisted of seven training drivers and one test driver. With `Optuna`, 1000 trials with different hyperparameter combinations chosen dynamically with TPE-sampling was carried out. The hyperparameter search space used for the Random Forest classifier is specified in Table 3.3. The optimization study aimed to minimize the Mean Squared Error (MSE) of the predicted ORS level. For each LOOCV split in each trial, an average validation performance of the current hyperparameter setup was evaluated by fitting the model on the seven training drivers and testing on the remaining one. Hence, the study was optimized on the test data, which must be considered when reviewing the results.

The hyperparameter setup that achieved the lowest MSE on the ORS prediction was obtained. With this setup, the average impurity-based feature importance (Gini importance) could be calculated since the classifier was a Random Forest. The importance of a spatial eye blink feature was computed as the normalized total reduction of the criterion brought by that feature [6].

It is important to keep in mind that the result could not indicate the most important features for all different classifier types, hyperparameter setups, and sequence durations. Rather, it could merely indicate which of the 36 eye blink features contain the most information regarding the driver's sleepiness level with this specific setup.

3.4.3 Sleepiness from Temporal features

As noted in Section 1.2, AUs from the Facial Action Coding System (FACS) (see Section 2.3.4) have previously been utilized with success when it comes to classifying sleepiness in a study performed by Vural et al. [12]. The study utilized simulated driver scenarios for data collection and entirely different target labels rather than conducting a naturalistic driving experiment as in this case. Nevertheless, it is of high interest to research whether similar features are viable in this project for classifying sleepiness towards the ORS target labels. Moreover, in the introduction of Section 3.4, it was further explained that RNN-based models such as the LSTM might be able to classify sleepiness directly upon temporal data in the form of time-series, instead of squeezing the time-series into spatial handcrafted features as explained in Section 3.4.2.

Thus, from the different video sequences of each driver, 20 different temporal features were extracted using OpenFace 2.0. These included 17 different AUs defined in Table 2.4, as well as the eye gaze angles for each eye which could all be obtained directly from OpenFace 2.0. Additionally, the Eye Aspect Ratio (EAR) defined in Section 3.3.2.3 was calculated from eye landmark positions. In total, the video sequences are encoded as 20-dimensional time-series vectors of length equal to the number of frames for any chosen sequence duration.

The basic prediction model is implemented in `Pytorch` and consists of a number of stacked LSTM layers. From the topmost LSTM layer's last hidden state, there is a fully connected feed-forward layer with an output size equal to five (the number of ORS targets). A softmax layer has been added from the fully connected layer to calculate the probabilities of each target. Finally, the cross-entropy loss is computed by adding a final negative log likelihood loss (NLLL) layer. The Adam optimizer was used for training.

In Figure 3.8 an overview of the pipeline for this approach is illustrated.

The sequences were augmented as specified in Section 3.4.1.4. The data of the training drivers in each CV split were fed to the LSTM model with a batch size of 16. Gradient clipping of the optimizer was introduced with a max norm of ± 1 (calculated over all gradients) to avoid the explosion of gradients.

Unfortunately, the long training times of the model made it impossible to set up the experiment as specified in Section 3.4.1. Therefore, no unbiased estimations regarding the performance or generalization ability were obtained for this approach. Instead, we focused on determining whether the approach could even be viable. Was it possible for an LSTM model to classify sleepiness at all? If the model could not overfit and classify sleepiness on previously seen drivers, there would be no reason to classify sleepiness on unseen drivers, which is a more challenging task. There would also be no point in conducting the time-consuming experiment of finding the unbiased performance score and generalization ability of a model that would never work. Therefore, LOOCV for all eight drivers in the dataset was used for hyperparameter optimization.

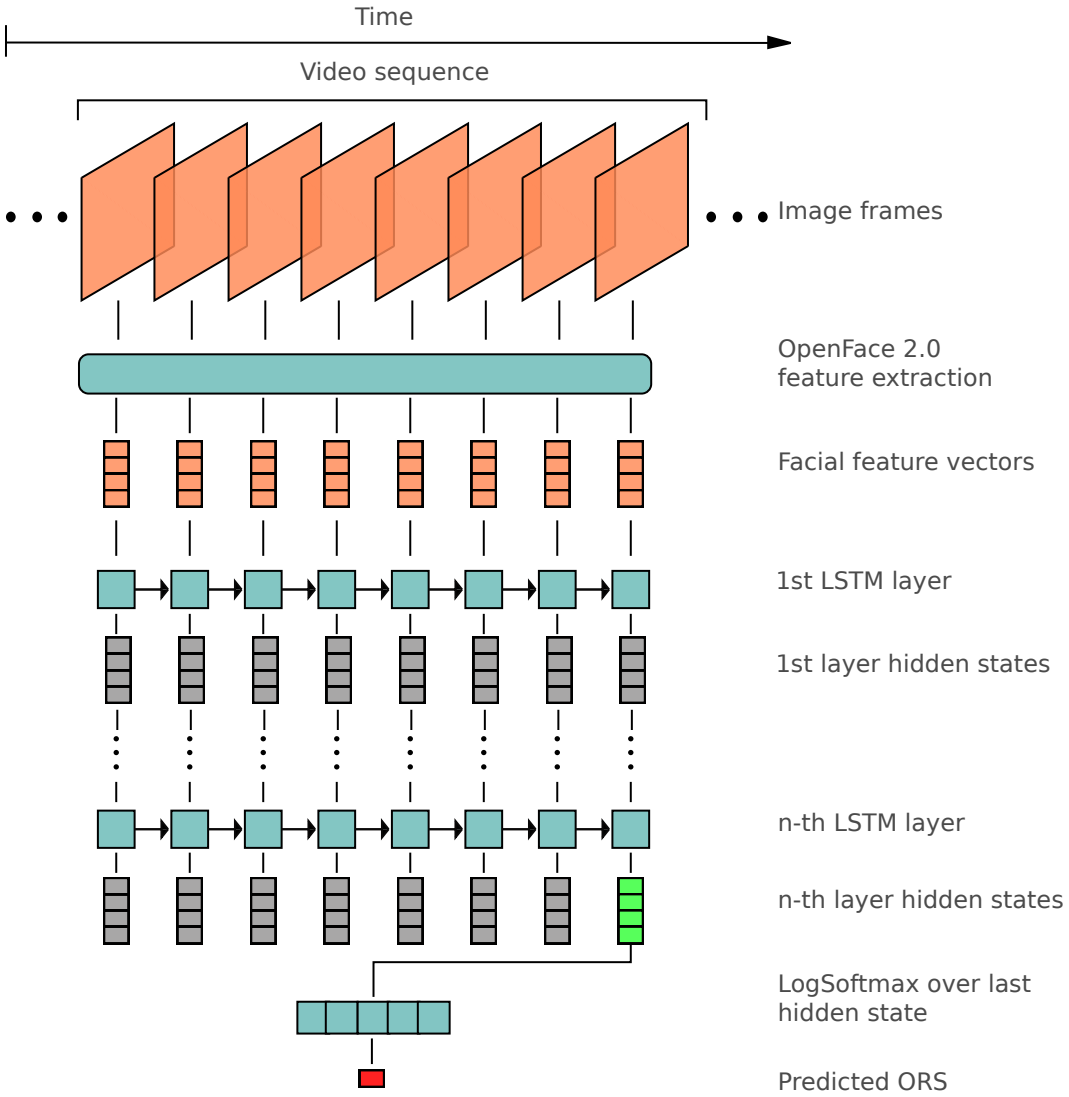

Figure 3.8: High-level illustration of pipeline for the *sleepiness from temporal features approach* using a stacked LSTM RNN. Note the difference from the *sleepiness from handcrafted eye blink features approach* illustrated in Figure 3.7.

Early stopping was utilized, as explained in Section 2.4.3.2. It was implemented by randomly splitting the data of the training drivers in each LOOCV split into two parts, 75 percent for training and 25 percent for validation. To make a distinction between the validation set used for early stopping and the held-out driver obtained in each LOOCV split, we refer to the held-out driver as the testing driver even though there was no nested cross-validation for this approach.

The cross-entropy loss was used as a validation score for the early stopping, with a patience of five epochs to determine when to terminate the training process. Additionally, during training and validation, the respective dataset splits were balanced using weighted random sampling such that the target labels were approximately

equally distributed.

Initial results indicated that the different models seemed to be overfitting on the training drivers. Therefore, additional runs were carried out to introduce dropout regularization with varying dropout probabilities. Dropout layers were added between stacked LSTM layers and before the fully connected layer. However, no indication of improved performance was observed, so the dropout probabilities were put to zero. Instead, the focus was placed within varying the LSTM architecture in the hopes of finding a model that would produce more promising results with less overfitting.

The training time seemed to increase approximately linearly with increased sequence duration. At this stage, the simplest evaluated model, a 1-layer LSTM with a hidden dimension of 64 for 30-second sequences, took approximately one hour to train and evaluate using an NVIDIA RTX 2060 Super GPU. Evaluating the same model with a sequence duration of 120 seconds instead took approximately 3.5 hours. One run using a 240-second sequence duration and 512 hidden dimensions for a 3-layer LSTM was conducted. However, it was terminated early after more than 8 hours of training. It also appeared to be overfitting and predicting randomly on testing drivers in the LOOCV loop. The time-consuming training and evaluation process severely limited exploring a large search space of hyperparameter combinations and larger network architectures, especially at longer sequence durations. In order to obtain interpretable results within the given timeframe, it was apparent that the experiment had to be downscaled.

As described in Section 2.4.2, LSTMs have been shown to possess limited ability in retaining a memory over sequences of magnitudes larger than 100 time steps in practice, even though they have unlimited memory in theory. The shortest sequence of our choosing at this point, 30 seconds sampled at 30 FPS, results in input sequences of 900 discrete time steps. Therefore, it was decided that it would be enough to test the 30-second sequence duration against the 120-second one to obtain initial results and indications of performance of varying sequence durations. Preferably, shorter sequence durations would have been interesting to consider, but as explained in Section 3.4.1.3 shorter sequence durations were problematic due to the uncertainty of labels in such cases. Therefore, to obtain results in a reasonable amount of time, a small hyperparameter search space of 16 combinations was constructed for which an exhaustive grid search was conducted. The search space is presented in Table 3.7. The results from these runs are presented in Section 4.2.2.

Table 3.7: LSTM hyperparameter search space of 16 different combinations.

Hyperparameter	Possible Values	Explanation
sequence_length	$k \in \{30, 120\}$	Input sequence duration in seconds. Multiply by 30 FPS for actual length of LSTM input.
hidden_dim	$k \in \{64, 512\}$	The number of features in the LSTM's hidden state.
stacked_layers	$k \in \{1, 2\}$	The number of recurrent layers. If k $>$ 1 a *stacked LSTM* is obtained.
learning_rate	$k \in \{0.001, 0.01\}$	Controls the size of each gradient descent step.

4

Results

This chapter describes the results from the eye state and sleepiness prediction. The inter-annotator agreement results are presented in Section 4.1, Eye State Classification Results. Furthermore, Section 4.2, Sleepiness Prediction Results, is split into two parts: sleepiness from *handcrafted eye blink features* and sleepiness from *temporal features*.

4.1 Eye State Classification Results

This section starts with the inter-annotator agreement results presenting how the annotators differ in their annotation technique. Next, the optimization study results are presented, leading to an optimal eye state classification model. Finally, the eye state classification results with decreasing percentages of the data are presented to learn how much annotated data is required.

4.1.1 Inter-Annotator Agreement Results

The inter-annotator agreement study results are presented in Figure 4.1. The diagonal elements represent annotations where both Casper and Anton annotated the same target. Off-diagonal elements are annotations where Casper and Anton disagreed. For example, for the frames where Anton annotates partially closed eye states, Casper disagrees and annotates 17 of these as closed and 10 as open.

Table 4.1 presents the precision, recall, F1-score, and support for each eye state class, where Casper's annotations are seen as true, and Anton's values are seen as predicted values. Note that there is a significant imbalance between the classes. The inter-annotation differs noticeably, especially on the partially closed eye states.

4.1.2 Optimization Study Results

The `Optuna` optimization study aimed to help determine the optimal size of the moving window W, which feature to choose, and the hyperparameters of the classifier.

In Figure 4.2, a slice plot of the trials run with different moving window sizes is presented. It is important to note that the F1-score also depends on many other hyperparameters, but this figure indicates how the moving window size affects the

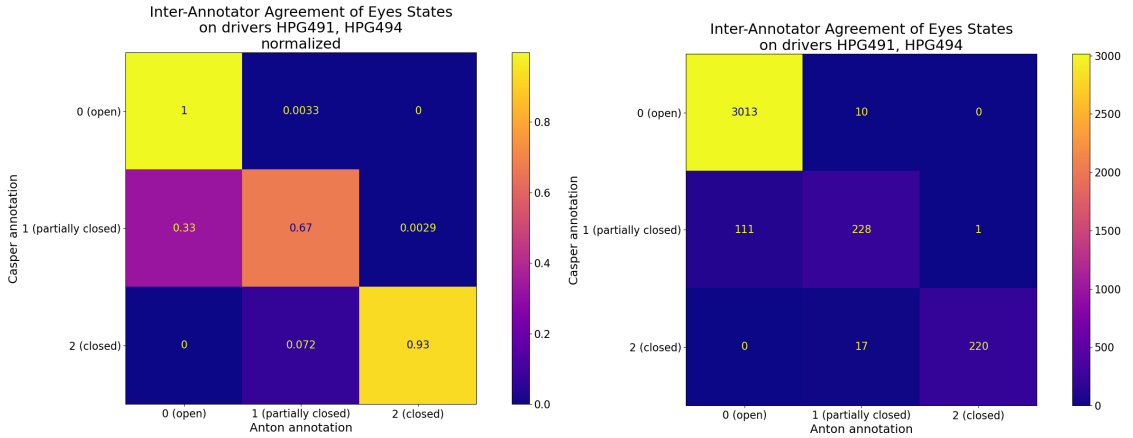

Figure 4.1: Inter-annotator agreement study between the authors Casper Lindberg and Anton Claesson on drivers HPG491 and HPG494. To the left, the normalized confusion matrix is presented.

Table 4.1: Classification report of the inter-annotator agreement study between the authors Casper and Anton on drivers HPG491 and HPG494. Casper's annotation is in this study considered as the true values and Anton's annotation is considered as the predicted values. The support is the total number of eye state annotations annotated by Casper of each class (0, 1, or 2).

Eye State	Precision	Recall	F1-Score	Support
0 (open)	0.96	1.00	0.98	3023
1 (partially closed)	0.89	0.67	0.77	340
2 (closed)	1.00	0.93	0.96	237

performance. The trials that achieved the highest weighted F1-score had a moving window size $W = 7$ frames. However, there is no significant difference in weighted F1-score using between 3 - 10 frames. By using fewer frames, the complexity of the model decreases. Hence, following Occam's razor principle, a suitable choice of the moving window size would be $W = 3$ frames.

Additionally to selecting a suitable moving window size, feature selection was performed by inference based on the results. Figure 4.3 shows that the highest weighted F1-scores seem to have been obtained by trials utilizing all four of the available features. Thus, each feature might be significant, and they were all selected for usage in the final eye state classification model.

Next, in Figure 4.4, the parallel coordinate plot for the eye state classification optimization study is presented. Note that the moving window and the features (EAR, blink intensity, and gaze angles) are excluded from this plot for easier interpretability. In this plot, the lines represent different trials, in total 150. Starting from left to right, the vertical axis on the left is the weighted F1-score for the eye state classification.

The bootstrap hyperparameter is considered next. The majority of the trials with

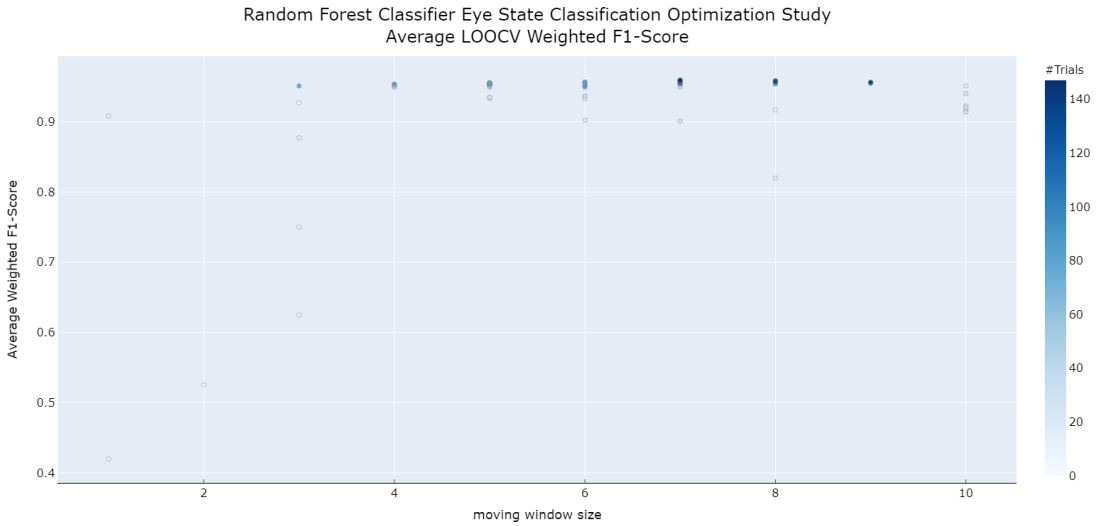

Figure 4.2: Slice plot for the Random Forest classifier from the `Optuna` study with the goal of maximizing the weighted F1-score for eye state classifications. The moving window size varies from 1 to 10.

a high weighted F1-score seem to disable the use of bootstrap. The max_depth parameter seems to give good results in the whole span from 2 - 100. Moreover, the max_features hyperparameter seems to have about equal performance with the auto and sqrt setting. For the min_samples_leaf hyperparameter, values of 1 or 2 seem to give good results. The min_samples_split parameter gives good results in the whole range from 2 - 20. Finally, the n_trees hyperparameter (the number of estimators) seems to give the best results in the interval 10 - 300.

By following the Occam's razor principle of prioritizing a simple model, which is also faster, the following parameter setup was used:

moving window size: 3, all features used (EAR, blink intensity, gaze angle x, and gaze angle y), 'n_trees': 20, 'max_depth': 17, 'bootstrap': false, 'max_features': auto, 'min_samples_leaf': 1, 'min_samples_split': 2

This resulted in the confusion matrices presented in Figure 4.5 and the classification report presented in Table 4.2. Note that the average weighted F1-score is slightly less than the best model in the F1-score optimization study presented in Figure 4.4 which had an average weighted F1-score of 0.9594 compared to 0.9496. However, the performance of the model is highly affected by the annotation, as seen in the inter-annotator agreement presented in Figure 4.1. Hence, the simpler model is selected. The LOOCV weighted average F1-score per driver is presented in Figure 4.6. Note that the weighted average F1-score varies slightly between the different drivers, with the worst performance on HPG489 and the best performance on HPG488. In Figure 4.6 one can also see the performance for each within-subject eye state classification model. Note that these models are of the same type as the LOOCV model but fitted on data only from one driver. The models are trained on each driver individually with 70/30 train/test splits. On average, the performance of

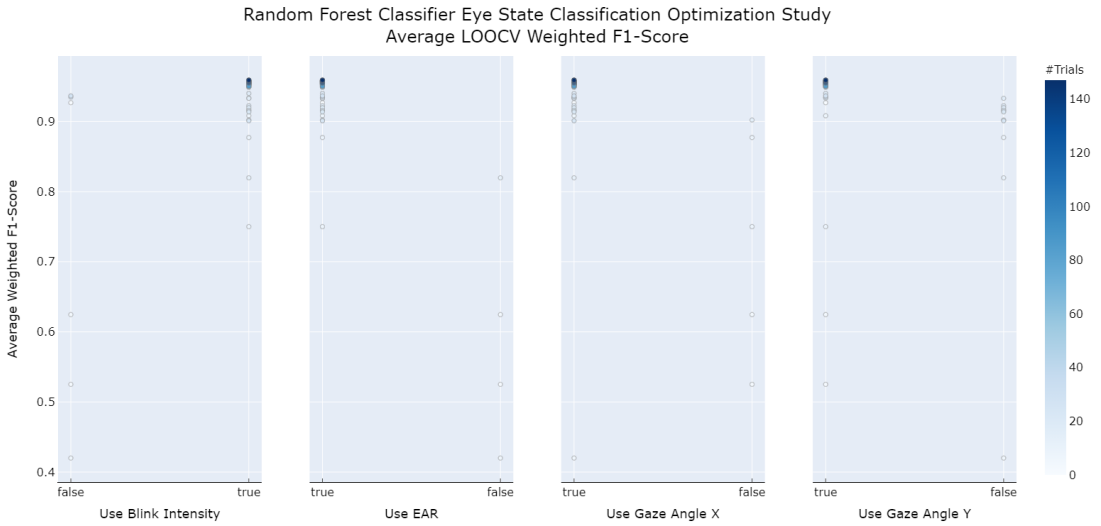

Figure 4.3: Slice plot for the Random Forest classifier from the `Optuna` study with the goal of maximizing the weighted F1-score for eye state classifications. The features investigated are EAR, blink intensity, gaze angle x and gaze angle y.

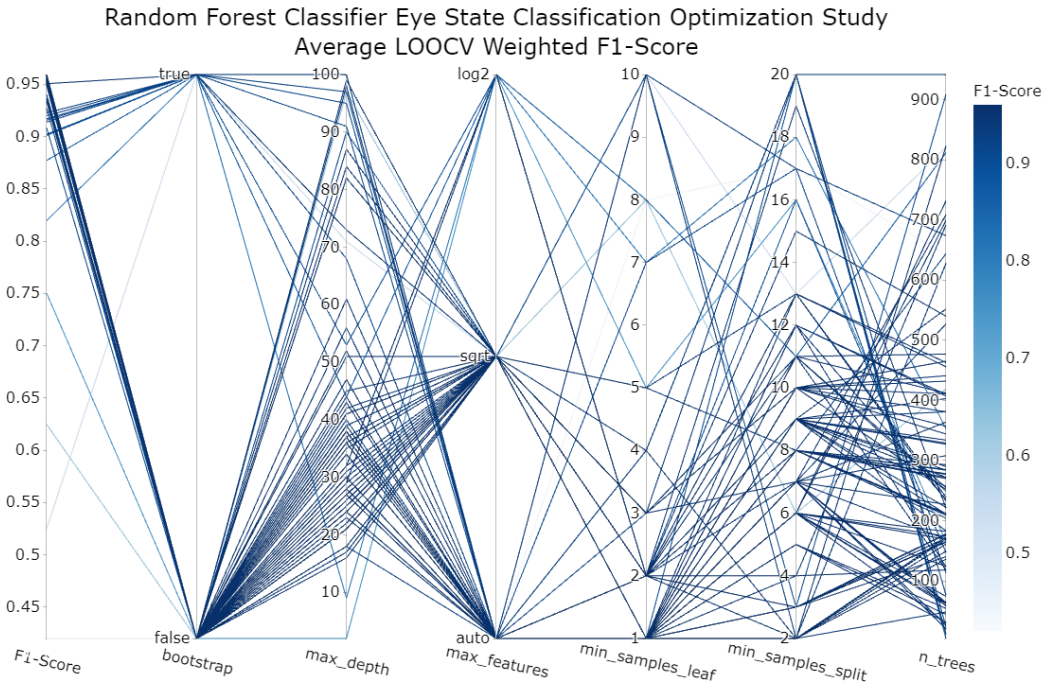

Figure 4.4: Parallel coordinate plot of 150 trials of Random Forest classifiers. The objective value on the vertical axis to the left is the average weighted F1-score. One line represents one trial.

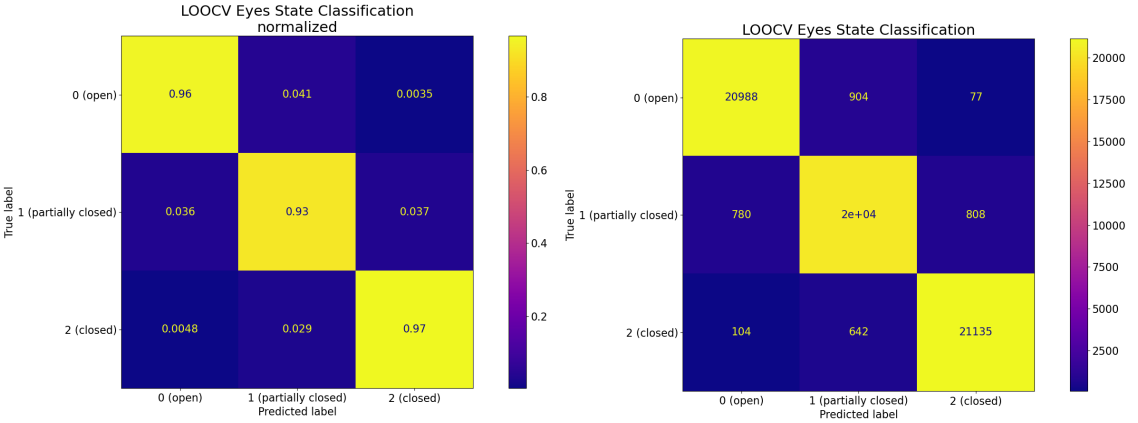

Figure 4.5: LOOCV eye state classification results. To the left, the normalized confusion matrix is presented.

Table 4.2: Classification report of the LOOCV eye state classification. The support is the total number of eye state instances of each class (0, 1, or 2).

Eye State	Precision	Recall	F1-Score	Support
0 (open)	0.96	0.96	0.96	21969
1 (partially closed)	0.93	0.93	0.93	21926
2 (closed)	0.96	0.97	0.96	21881
weighted average	0.950	0.950	0.9496	-

the within-subject models is about 1 % higher than the general LOOCV model. However, for 3 out of 8 drivers, the performance is better when using the general LOOCV model.

This simpler model was fitted to all available data. However, to sanity check that the performance did not drop by adding the final driver, 1/8:th of the data was randomly sampled as a test set. The performance of the sanity check is presented in Table 4.3. The performance is slightly higher than in the LOOCV study as expected.

Next, the selected eye state classification model was trained on parts of the original data. The results are presented in Figure 4.7. Using all data leads to the best overall

Table 4.3: Classification report of the train-test split sanity check of the eye state classification model with a moving window size $W = 3$ frames, all features and the hyperparameters specified in the list above. The support is the total number of eye state instances of each class (0, 1, or 2).

Eye State	Precision	Recall	F1-Score	Support
0 (open)	0.96	0.96	0.96	1308
1 (partially closed)	0.94	0.93	0.93	1305
2 (closed)	0.96	0.97	0.97	1302
weighted average	0.954	0.954	0.9539	-

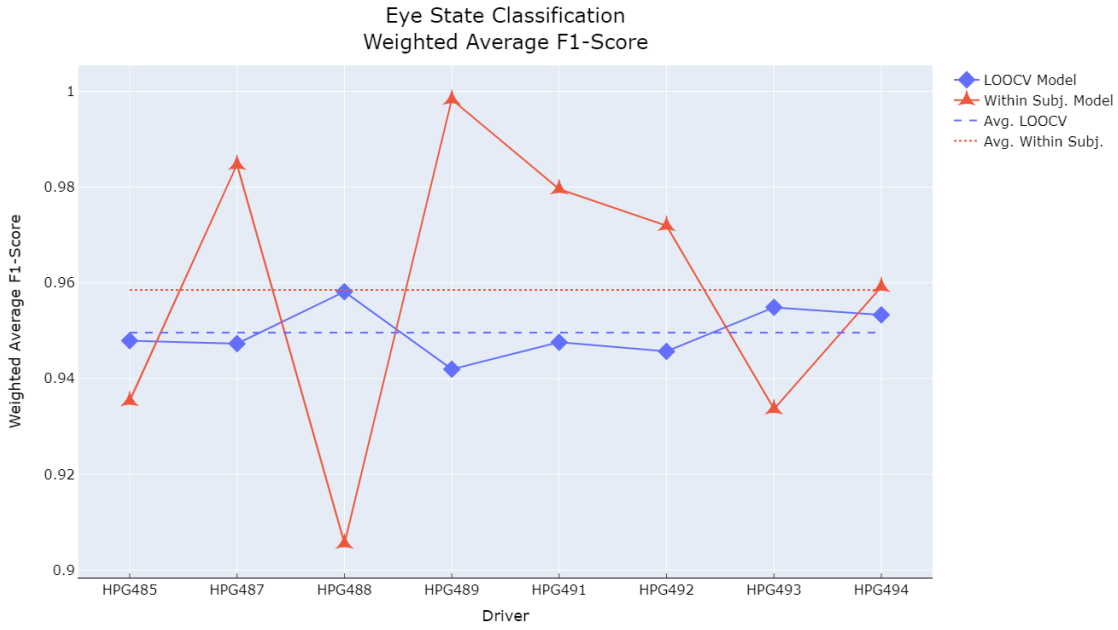

Figure 4.6: LOOCV and within-subject eye state classification weighted average F1-score per driver. The figure also shows the average across all drivers for both the LOOCV model (dashed line) and the within-subject models (dotted line). The within-subject models are trained and tested on the same driver with a 70/30 train/test split.

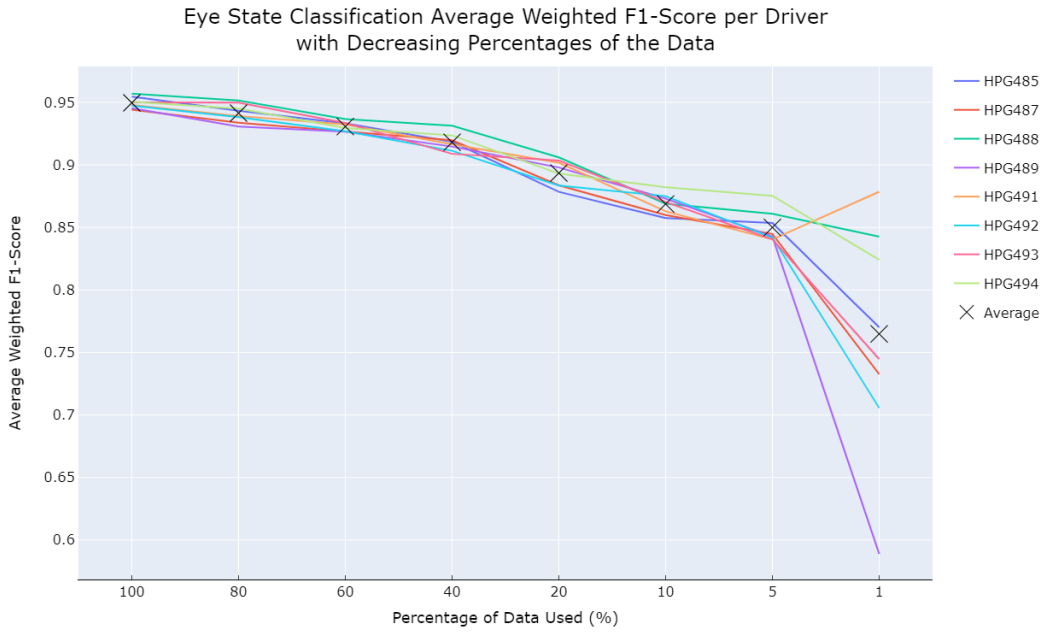

Figure 4.7: Average weighted F1-score per driver with different amounts of the annotated data (see Table 3.1), from 100 % to the left to 1 % to the right. The thin crosses represents the average across all drivers using the specified percentage of the data.

weighted average F1-score. The figure shows how much the performance drops when the amount of data is decreased. The most significant drop happens from 5 % to 1 % of the data, especially for HPG489 who has the least amount of annotated data. 1 % of HPG489's annotated data corresponds to 63 frames.

4.2 Sleepiness Prediction Results

This section presents the results of classifying the driver ORS level using two different approaches. The first approach is based upon *handcrafted eye blink features* with a Random Forest classifier, and the second one upon *temporal features* with a Long Short-Term Memory (LSTM) classifier. The results of the baseline predictors are presented in Figure 4.8. The constant baseline predictor classifies the median ORS level (2) with an average MSE of 1.72. The uniform random baseline model gets an average MSE of 3.70.

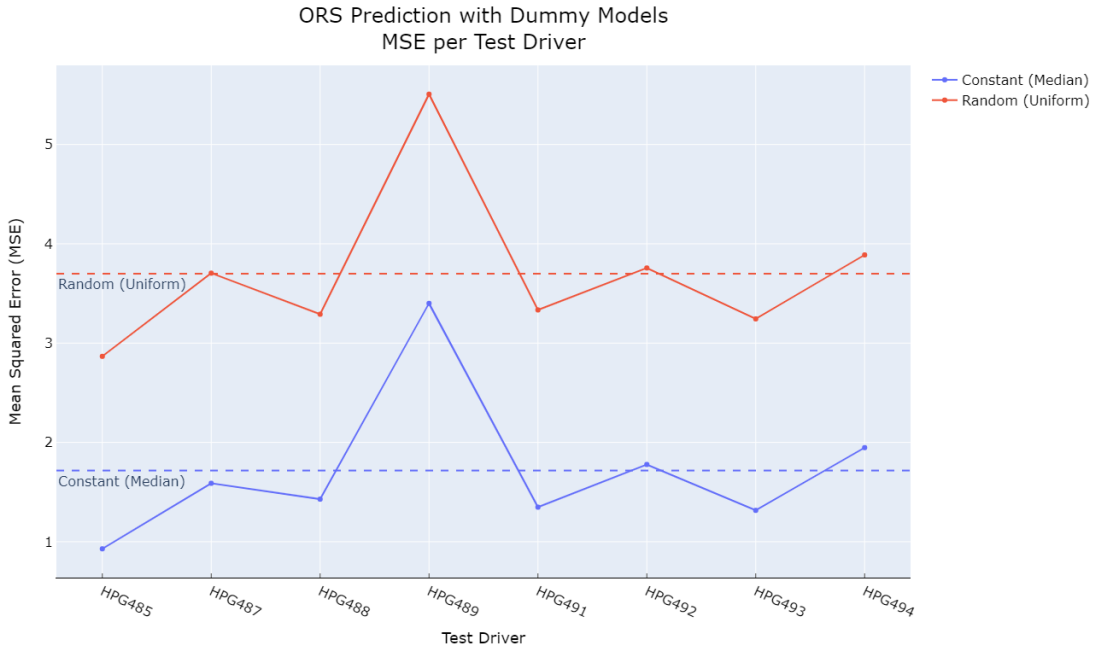

Figure 4.8: Results from the baseline predictors.

4.2.1 Sleepiness from Handcrafted Eye Blink Features Results

This section presents the results from the sleepiness prediction task using *handcrafted eye blink features* with a Random Forest model. The results are based upon the methodology presented in Section 3.4.1 and 3.4.2. Recalling the four evaluation methods defined, the results from Method A, Method B, Method C, and Method D are presented in chronological order.

4.2.1.1 Method A Results

Table 4.4 presents the parameter setups [S1, . . . , S8] for the Random Forest classifier. Recall that these setups were obtained from the eight different `Optuna` optimization studies of 250 trials each. As specified in the table caption and further explained in Section 3.4.1, the *setup loss* is the average LOOCV MSE using the specified parameter setup for the classifier, when excluding the driver noted column-wise

from the dataset. The results from Method A is also presented in Figure 4.9. The results are presented as boxplots to illustrate the variance of the ten different runs performed to reduce the impact of the random balanced sampling.

Parameter set S5, calculated when driver HPG491 was excluded, had the smallest setup loss, 0.901, and the lowest variance, see Figure 4.9. The largest loss was obtained when excluding driver HPG493 for setup S7, which also had the highest variance. Note that having the smallest setup loss does not guarantee that the corresponding setup is the most suitable since two primary reasons cause a low loss. Either, the parameter setup works exceedingly well for sleepiness prediction on the drivers, but it is also possible that the excluded driver is particularly challenging. If the latter is true, the average LOOCV MSE on the remaining (easier) drivers decreases compared to when the difficult driver is included in the LOOCV procedure.

Finally, also note that all setups $[S1, \ldots, S8]$ results in quite complex Random Forest models with many estimators and/or a high maximum depth.

Table 4.4: Parameter setups $[S1, \ldots, S8]$ with the sequence duration in seconds and the hyperparameters for the Random Forest classifiers. The setup loss is the average LOOCV MSE for fitting a Random Forest classifier with the specified parameter setups S1 - S8, when excluding the driver in parenthesis (HPGXXX) from the dataset.

Setup→ Params↓	**S1** (HPG485)	**S2** (HPG487)	**S3** (HPG488)	**S4** (HPG489)	**S5** (HPG491)	**S6** (HPG492)	**S7** (HPG493)	**S8** (HPG494)
sequence length (s)	180	180	180	180	120	180	180	180
n_ estimators	809	251	996	723	993	654	665	379
max_ depth	83	78	7	16	65	98	46	49
min_ samples _split	20	15	11	4	13	6	9	15
min_ samples _leaf	9	10	3	5	5	6	3	8
max_ features	log2	auto	sqrt	auto	auto	log2	auto	log2
bootstrap	True	False	False	True	False	True	True	True
Setup Loss	1.684	1.262	1.268	1.409	0.901	0.982	1.861	1.010

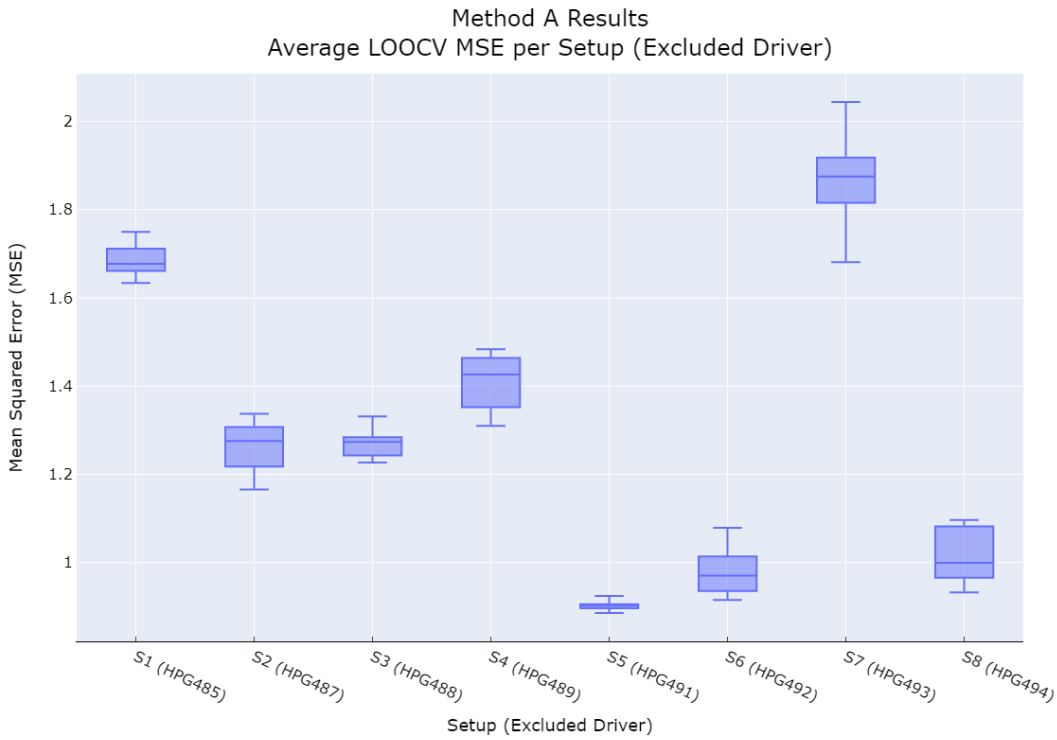

Figure 4.9: Results from Method A: the average LOOCV MSE of all drivers excluding the driver on the x-axis with the hyperparameter setup [S1, . . . , S8] respectively, read more in Section 3.4.1.

4.2.1.2 Method B Results

Recall that for this method, eight different Random Forest classifiers, one for each setup [S1, . . . , S8], were fitted on all drivers except the excluded test driver. Then, a test MSE was calculated on the excluded driver corresponding to each setup.

The results are presented in Figure 4.10 for the legend name *Method B* with an average MSE of 1.21. As mentioned in Section 3.4.2, this would correspond to a fair approximation of the general performance of an ensemble classifier consisting of the eight different Random Forest models.

Moreover, in the figure, the results are presented as boxplots to illustrate the variance of the ten different runs performed to reduce the impact of the random balanced sampling. In particular, note that the MSE differs significantly between different test drivers: from ∼0.5 on HPG485 to ∼2.8 on HPG494. As assumed, the test variance is indeed significant. Additionally, similar difficulties arise as in Method A Results when analyzing the models; the MSE can either obtain a low value because the corresponding model is good, or because the ORS of the testing driver is easy to classify. Conversely, the MSE can be high because the model is inadequate at classifying sleepiness or because the sleepiness level of the testing driver is hard to classify.

This is also why Method A Results should be used for picking out the best hyperparameter setup, as the scores from this method are obtained from an average of seven drivers instead of just one. However, these scores are slightly overestimated from the hyperparameter optimization process, so it is also important to consider the test results obtained here in Method B Results to see the complete picture.

4.2.1.3 Method C Results

Since parameter setup S5 achieved the lowest *setup loss*, it was considered the best setup. A new LOOCV procedure across all drivers was carried out with S5, where seven drivers were used for fitting the model and the remaining one for testing in each split. The resulting MSE per driver and an average across all drivers of 1.32 is presented in Figure 4.10 with legend name *Method C*. The results are presented as boxplots to illustrate the variance of the ten different runs performed to reduce the impact of the random balanced sampling. Note that the MSE differs greatly between different drivers: from ∼0.25 on HPG489 to ∼3.2 on HPG488. Also note that the performance on HPG491 should be approximately the same for both Method B and Method C which seems to be the case according to Figure 4.10.

4.2.1.4 Method D Results

As explained in Section 3.4.2, a hyperparameter optimization study using LOOCV for all eight drivers was conducted in order to explore the feature importances and upper limit performance. As this meant there were no excluded drivers for which an unbiased testing score could be calculated for the fitted models, the results can merely indicate feature importance and performance.

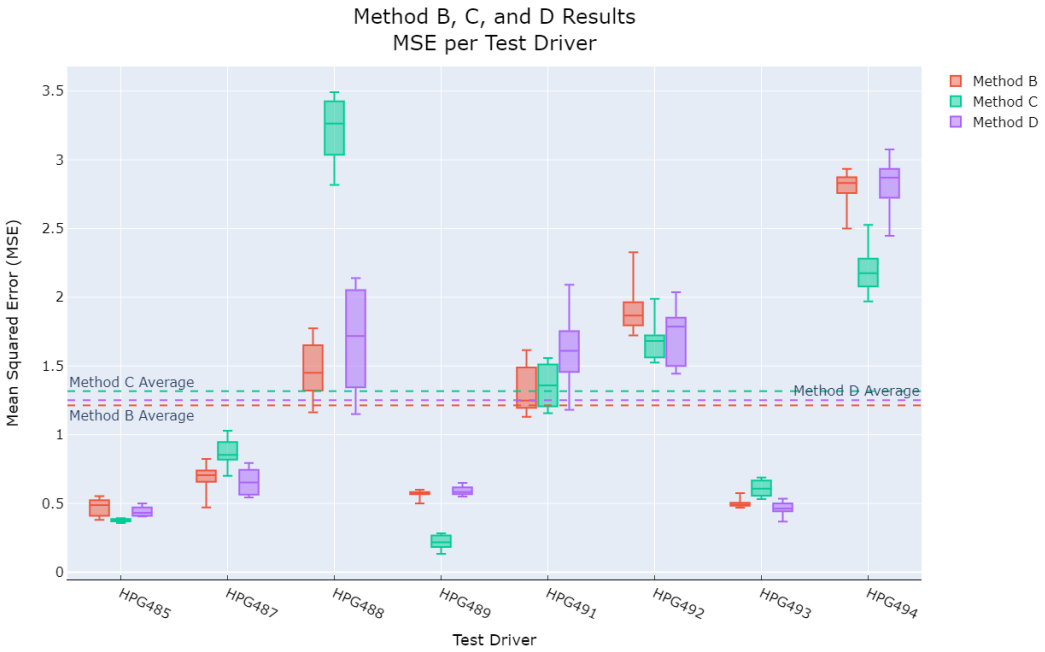

Figure 4.10: ORS prediction MSE test results per driver and per evaluation method Method B, Method C, and Method D read more in Section 3.4.1. Method B: unbiased test MSE on the driver specified on the x-axis fitted on the rest of the drivers with the best hyperparameter setups [S1, . . . , S8] per driver. Method C: the hyperparameter setup that achieved the lowest average LOOCV MSE from Method A, in this case from excluding HPG491, was considered as the final prediction model. This model was fitted on all drivers except the driver specified on the x-axis which it was tested on. Method C presents these test results. Method D: test MSE on the driver specified on the x-axis fitted on the rest of the drivers using a model that has been hyperparameter tuned on the test data.

The results of the sleepiness prediction `Optuna` hyperparameter study with 1000 trials using a Random Forest classifier is presented in Figure 4.11 and Figure 4.12. The model with best performance across all drivers, with an average MSE of 1.0532 during the `Optuna` study, had the following parameter setup (where sequence length is the sequence duration):

'sequence length': 180, 'n_estimators': 630, 'max_depth': 23, 'min_samples_split': 7, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'bootstrap': False.

Note that the low MSE is partly affected by the randomness of balanced random sampling performed to obtain an equal distribution of targets in the training set for each LOOCV split, as in this case, each of the 1000 trials was only run once. Recall that the other methods repeated the evaluation 10 times to compute an average and reduce the effect of the balanced sampling of the training data. A better approximation of the true average MSE for Method D is presented in Figure 4.10 which is equal to 1.25. The results are presented as boxplots to illustrate the variance of the ten different runs performed to reduce the impact of the random balanced

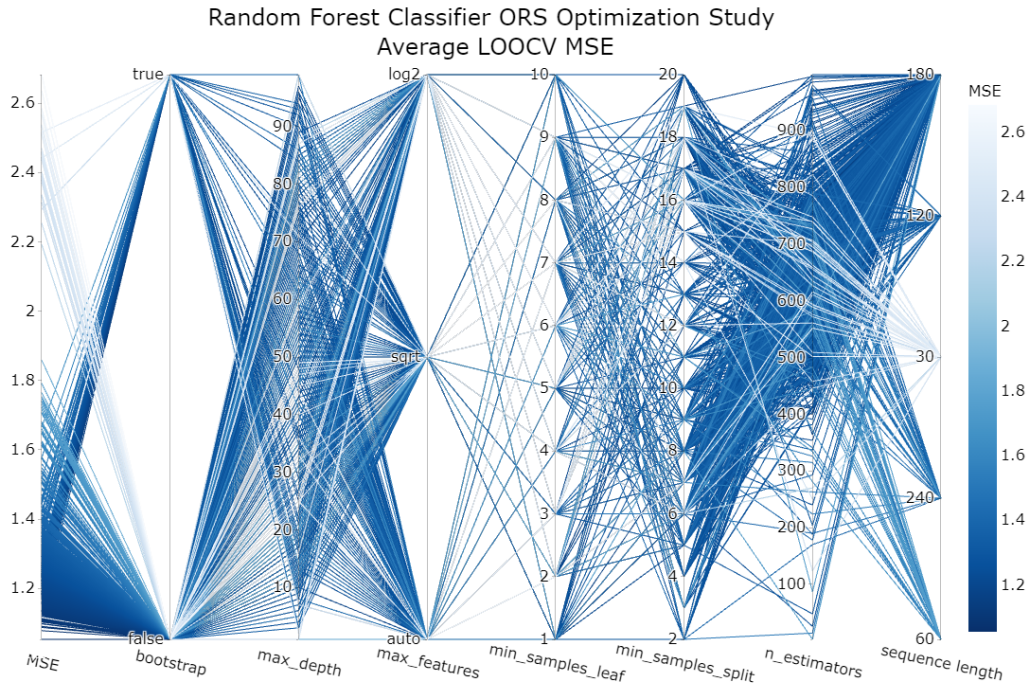

Figure 4.11: Parallel coordinate plot of 1000 trials of Random Forest classifiers. The objective value on the vertical axis to the left is the MSE. One line represents one trial.

sampling. Note that the average MSE for Method D is larger than for Method B but smaller than for Method C. Also, note that the worst performance is obtained when validating on HPG491 and HPG494.

From Figure 4.11 and Figure 4.12, observe that trials using longer sequence durations in general obtained a lower average MSE. As previously mentioned, TPE sampling chooses promising hyperparameter combinations depending on the result of previous trials, which is also why the `Optuna` study ran more trials with longer sequence durations. The best trials seem to be the ones with a sequence duration of 180 seconds.

The result of the model with the smallest MSE is presented as a confusion matrix in Figure 4.13 with a corresponding classification report in Table 4.5. From the confusion matrix it is clear that most observations are within a margin of ± 1 sleepiness levels with some exceptions.

Finally, the confusion matrices for each driver can be observed in Figure 4.14. Focusing on driver HPG494, one can see that the classifier seems to classify the median ORS level. On the other hand, for drivers HPG485, HPG487, HPG488, HPG489, HPG492, and HPG493, the model is moderately accurate within a tolerance of approximately ± 1 ORS levels. Recall that with an average MSE of 1.0532, this is expected.

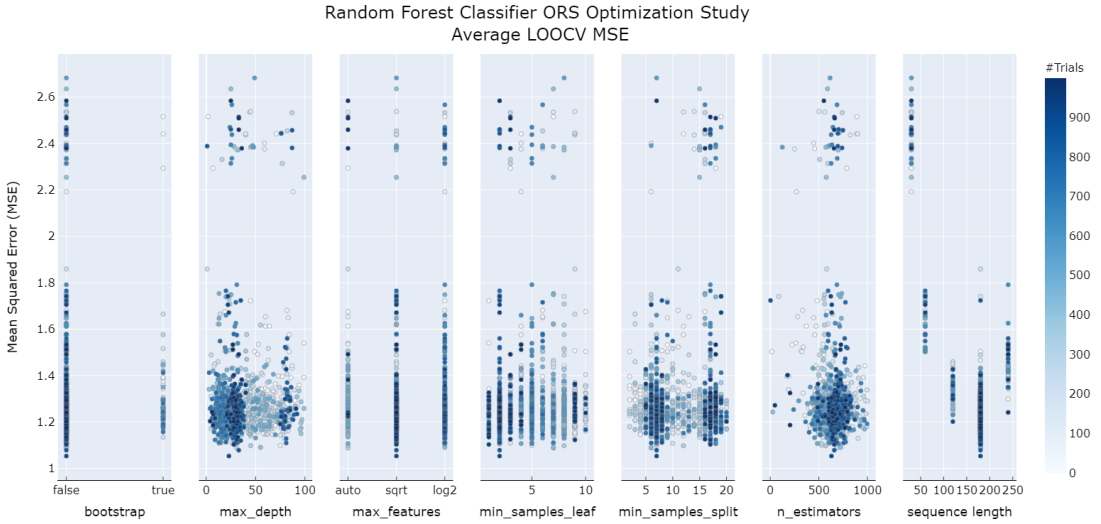

Figure 4.12: Slice plot for the from the `Optuna` study for the Random Forest classifier with the goal of minimizing the MSE for ORS sleepiness prediction.

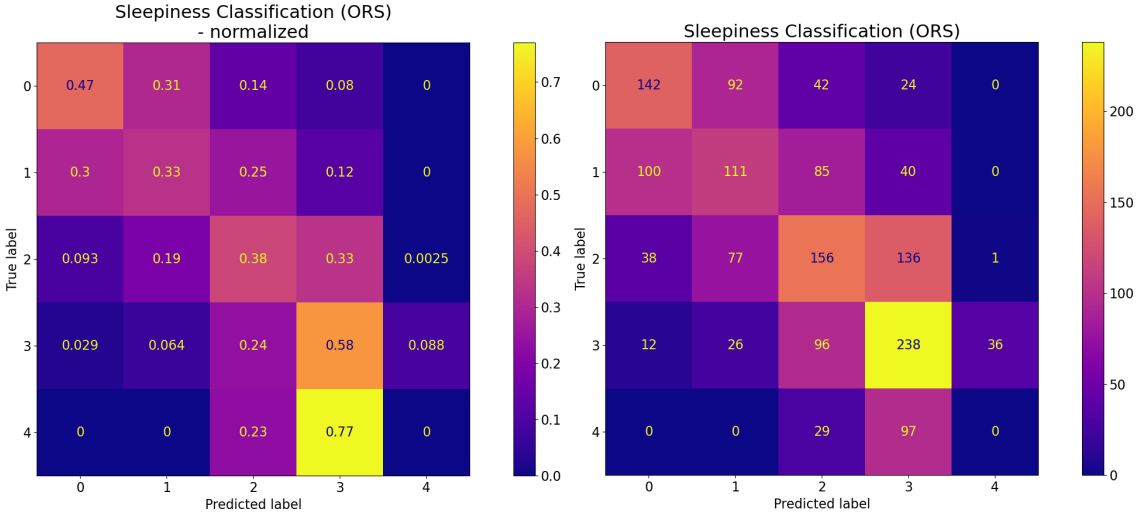

Figure 4.13: Confusion matrices (normalized and non-normalized) of the performance of the best (in terms of lowest MSE) Random Forest classifier found in the `Optuna` optimization study.

Table 4.5: ORS sleepiness classification report for the best Random Forest classifier showing the value per class and the weighted average of the precision, recall, F1-score, and the support for each ORS level.

ORS Sleepiness Level	Precision	Recall	F1-Score	Support
0	0.49	0.47	0.48	300
1	0.36	0.33	0.35	336
2	0.38	0.38	0.38	408
3	0.44	0.58	0.50	408
4	0.00	0.00	0.00	126
weighted average	0.384	0.410	0.394	-

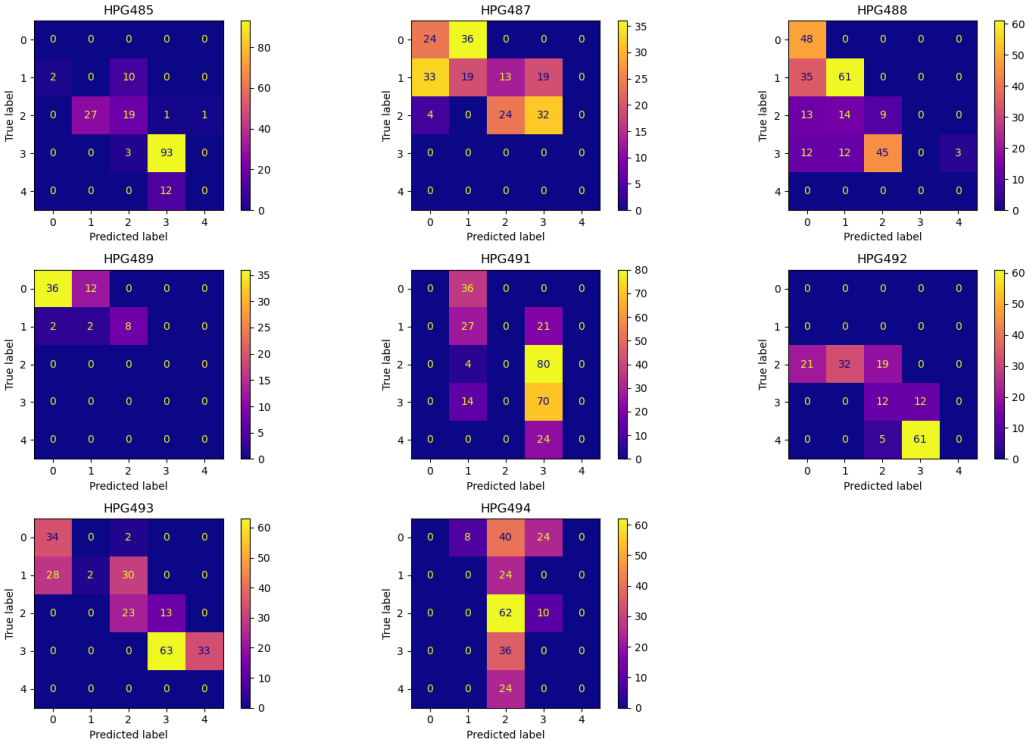

Figure 4.14: Confusion matrices for all drivers based on the best Random Forest classifier.

4.2.1.5 Feature Importance Results

Using the best hyperparameter setup from Method D, another LOOCV procedure was performed to evaluate the average feature importance of all eight fits. The result of the feature importance test with the Random Forest sleepiness classifier is presented in Figure 4.15. Note that this is a LOOCV test with all drivers. The top ten handcrafted eye blink features include:

1. The variance and average eyes opening and closing duration, (var_eyes_opening_duration, avg_eyes_closing_duration, var_eyes_closing_duration, avg_eyes_opening_duration).
2. The percentage of eyelid closure over time, where the eyes are partially closed (perclos).
3. The duration for which the eyes are fully closed (eyes_closed_duration).
4. The duration for which the eyes are partially closed (eyes_half_closed_duration).
5. The percentage of eyelid closure over time, where the eyes are partially closed (perclos_half_closed).
6. The longest duration of any eye closure in the sequence (max_blink_duration).
7. The number of blinks in the sequence, where the eyes are fully closed at least once (normal_blink_count).

For exact definitions, refer to Section 2.1.1.2. Note that the features *perclos* and *eyes_closed_duration* as well as *perclos_half_closed* and *eyes_half_closed_duration* are 100% correlated. Therefore, it would have been preferable to exclude one of each pair in the study.

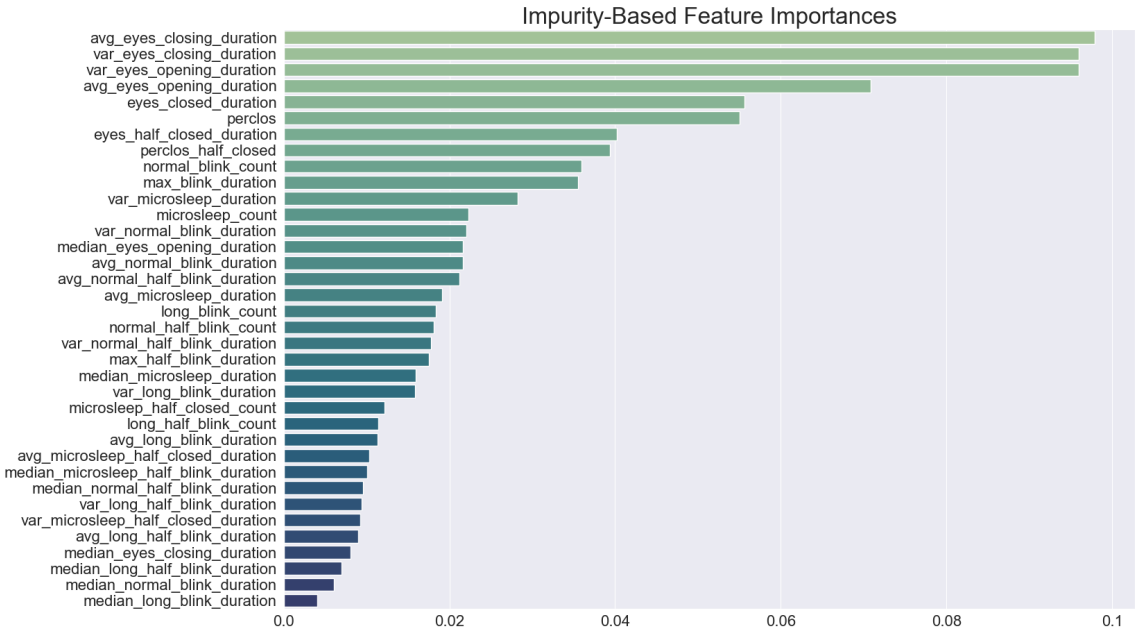

Figure 4.15: Feature importance results from the best trial with the Random Forest classifier.

4.2.1.6 Generalization Results

The effect of adding or removing training drivers (generalization ability) is measured based on parameter setup S5, the same setup used in Method C. Recall that this setup was chosen based on the smallest *setup loss* obtained in Method A, for which the results can be seen in Table 4.4.

The results are presented in Figure 4.16. The graph shows how the average MSE decreases as the number of drivers in the training set increases. The lowest average MSE is achieved from the LOOCV with seven drivers in the training set and one driver in the test set. The worst average MSE is achieved from the L70CV (Leave-7-Out Cross-Validation) with one training driver and seven test drivers. It is important to keep in mind that the size of the test set decreases as the number of training drivers increases. In summary, this figure indicates that having more drivers in the training set increases the general ORS prediction performance.

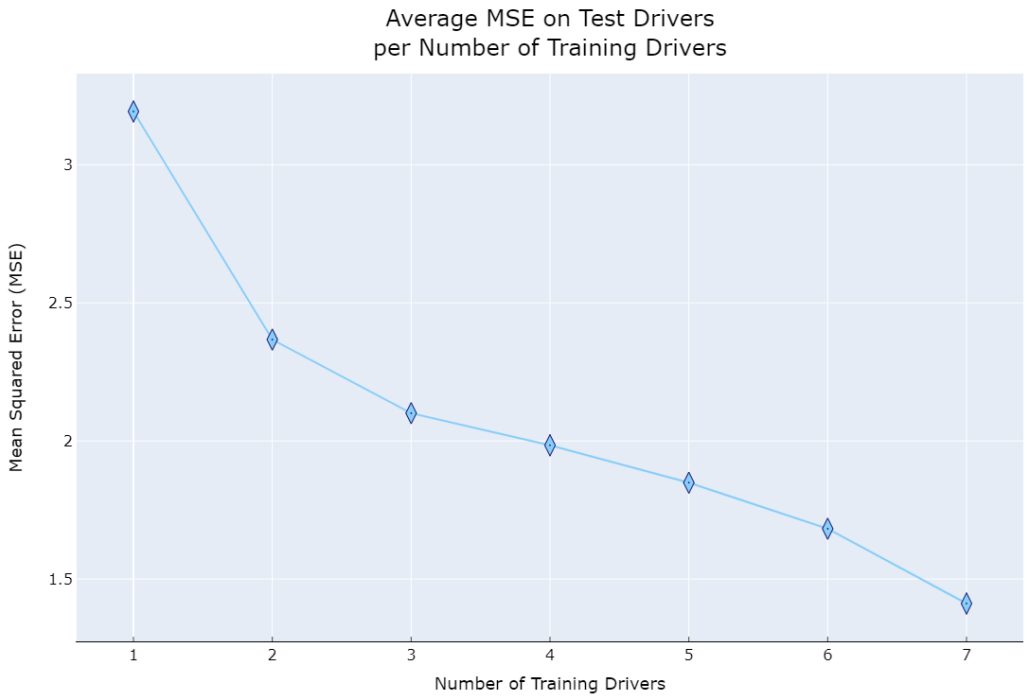

Figure 4.16: Average ORS prediction MSE per number of training drivers. The average MSE is the average over all possible LXOCV combinations where X indicates the number of test drivers (eight minus the number of training drivers).

4.2.2 Sleepiness From Temporal Features Results

The long-drawn training times required for this approach made it impossible to conduct the experiment like in the *sleepiness from handcrafted features approach* presented in Section 3.4.2. Instead, the objective was to determine whether the approach showed promising initial results or not. This is described in further detail in Section 3.4.3.

Recall that an optimization study using `Optuna` was conducted with one trial for each of the 16 different hyperparameter combinations. Each trial used a LOOCV procedure across all eight drivers in order to calculate an average MSE.

Unfortunately, none of the trials obtained a test performance close to what would be deemed acceptable in a real-world application. In fact, no parameter setup showed clear indications of outperforming any other. The model predicted quite randomly for all trials, but only when it was presented with unseen drivers. Judging by the results, it seems as the model quite accurately manages to learn how to predict the sleepiness of drivers it has previously seen. Unfortunately, such a model is unusable for actual real-world implementation as it would require annotated data for every new Volvo Cars end customer. However, it remains a crucial step towards obtaining a generalized model.

Nevertheless, the aforementioned result can be observed by inspection of the curves in Figure 4.17 and Figure 4.20. The former figure illustrates a trial with a sequence duration of 30 seconds, and the latter a trial with a sequence duration of 120 seconds. Each line corresponds to one LOOCV split and LSTM training process where one driver (indicated by legend color and name) was held out for testing and the other drivers were used to train the model. The filled lines show the *validation MSE* obtained on the validation set (25% of the training data) per epoch and split. Each model was evaluated based on the cross-entropy loss on the validation set. Consequently, the MSE validation curves are not strictly decreasing, since when a model finishes an epoch, it might obtain a lower cross-entropy validation loss, even if the validation MSE increases. The dotted colored lines point to the test score obtained on the specific held-out driver for the respective splits once training was finished. Also, note that each LSTM training procedure was terminated at different points since early stopping with a patience of five trials was utilized. Since the validation set was built from 25% of mixed training data in each split, it included the same drivers as in the training set.

The validation MSE curves are steadily decreasing down to an average slightly larger than one. The decreasing validation MSE indicates that the model learns to classify the sleepiness level when presented with new sequences from previously seen drivers, with an error slightly larger than ± 1 targets. The error should be observed in relation to the fact that there are five different ORS levels.

However, once the model is tested on new drivers, it fails miserably, indicated by the large jumps in the graphs. Inspection of the confusion matrices yields the same conclusion.

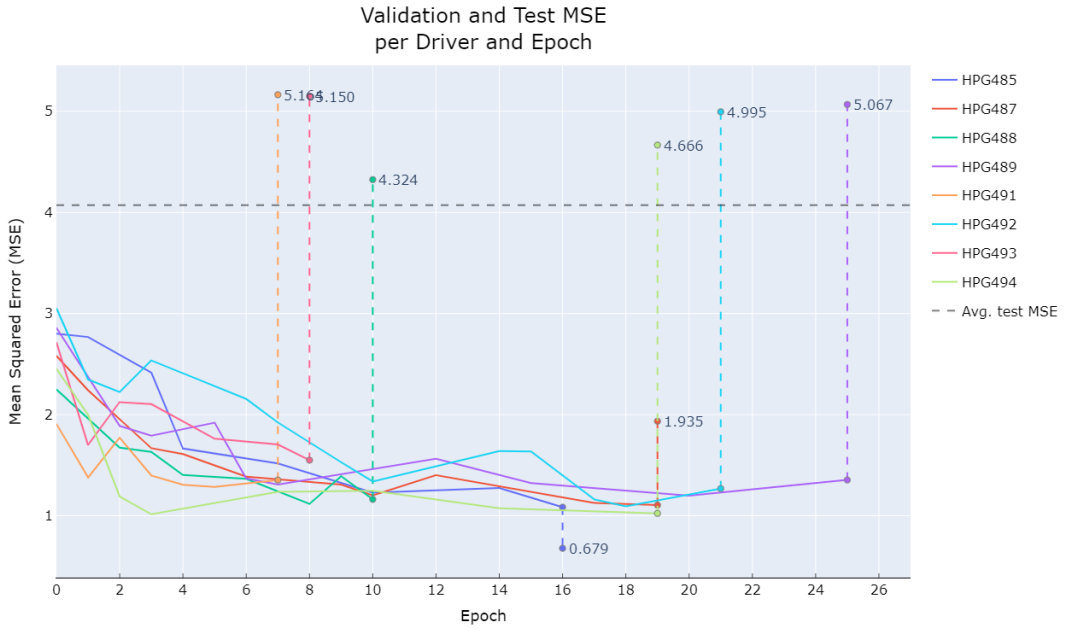

Figure 4.17: Valdiation and test MSE for an LSTM model with sequence duration 30 seconds, 1 LSTM layer with 64 in hidden dimension size and a learning rate of 0.001. The filled lines correspond to validation MSE for the best so-far model instance according to the cross-entropy loss, which is connected with a dotted line to the corresponding test MSE for The corresponding confusion matrix for total predictions is presented in Figure 4.18, and confusion matrices per driver are found in Figure 4.19.

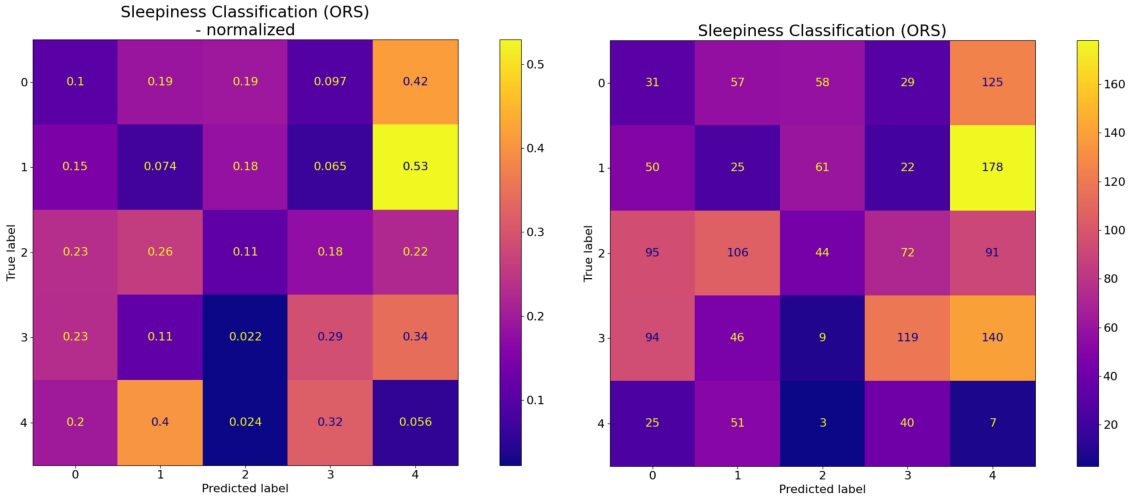

Figure 4.18: Confusion matrix of total test predictions for the evaluation presented in Figure 4.17.

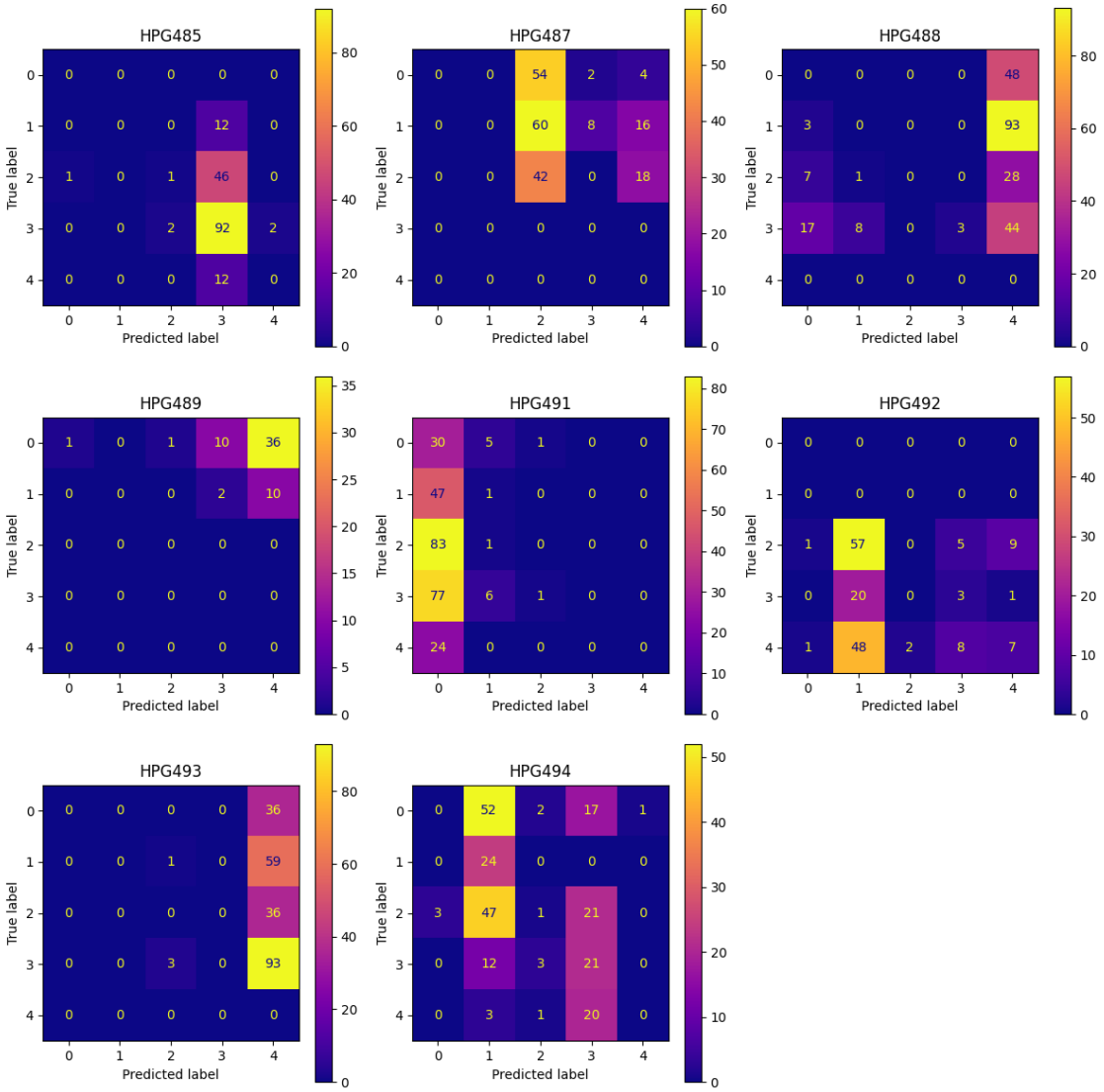

Figure 4.19: Confusion matrices of test predictions per driver for the evaluation presented in Figure 4.17.

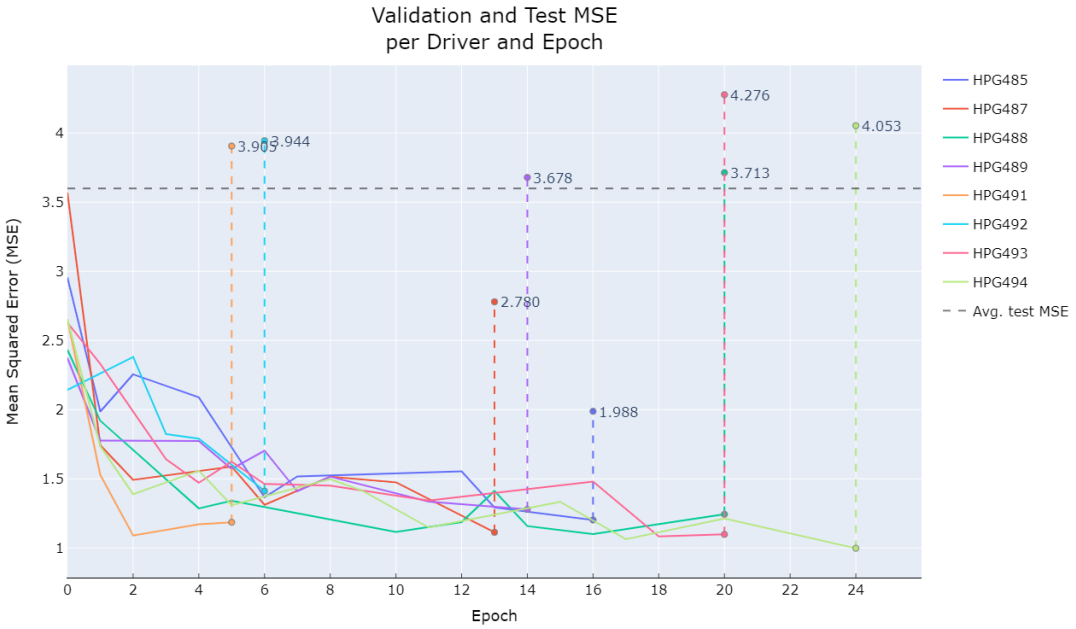

Figure 4.20: Valdiation and test MSE for an LSTM model with sequence duration 120 seconds, 1 LSTM layer with 64 in hidden dimension size and a learning rate of 0.001. The filled lines correspond to validation MSE for the best so-far model instance according to the cross-entropy loss, which is connected with a dotted line to the corresponding test MSE for each held-out driver in every CV split. The corresponding confusion matrix for total predictions is presented in Figure 4.21, and confusion matrices per driver are found in Figure 4.22.

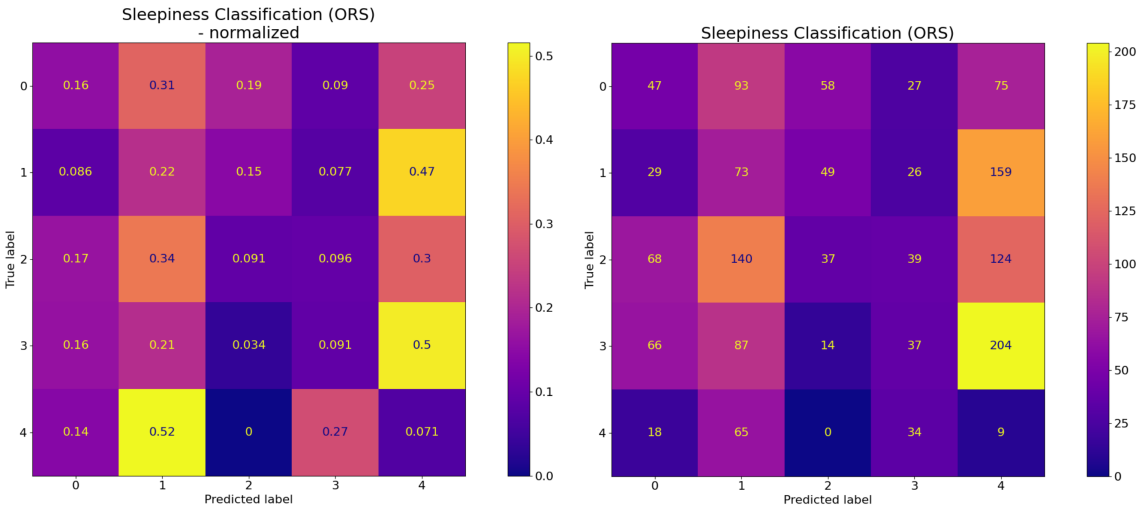

Figure 4.21: Confusion matrix of total test predictions for the evaluation presented in Figure 4.20.

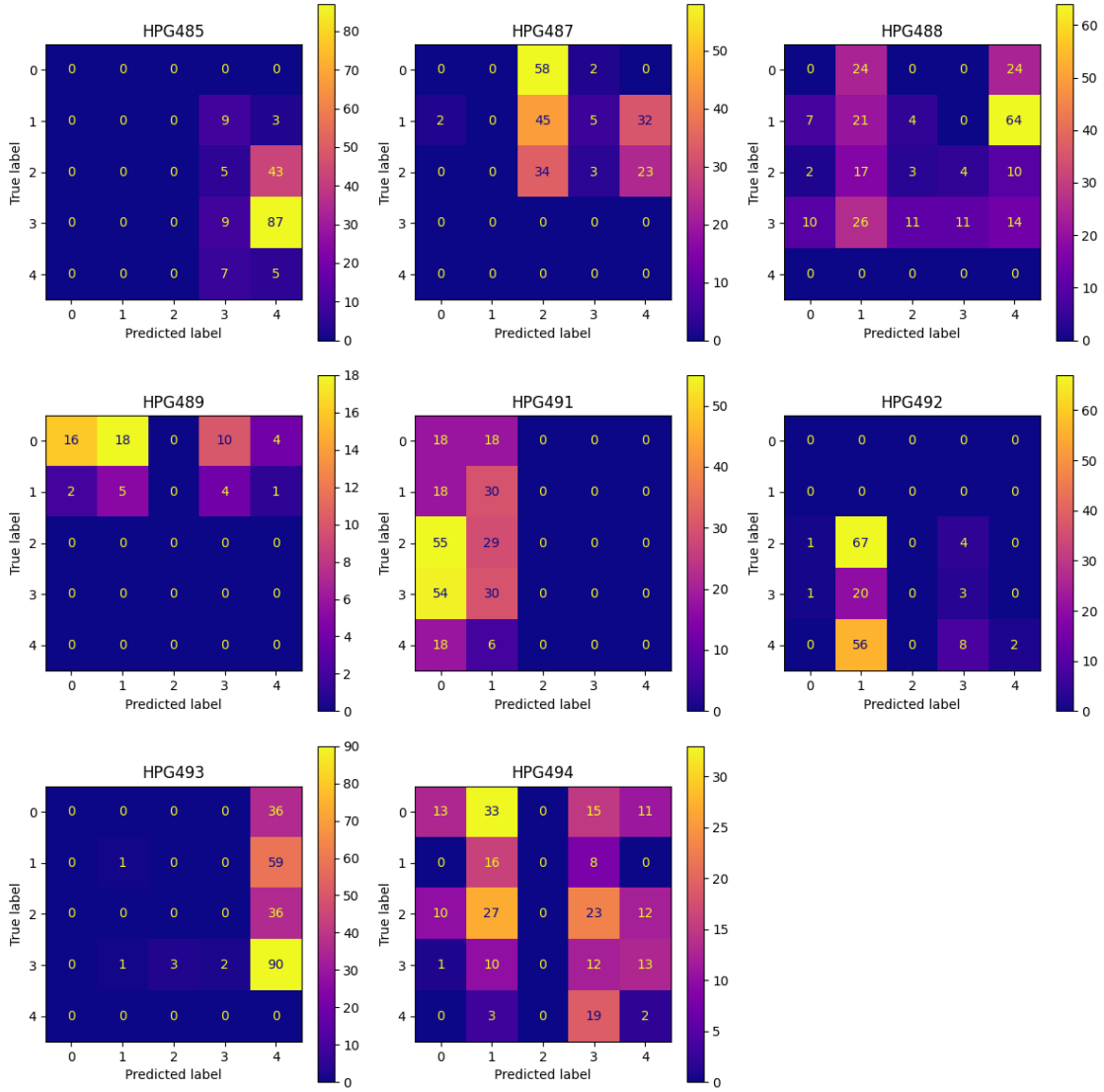

Figure 4.22: Confusion matrices of test predictions per driver for the evaluation presented in Figure 4.20.

4.2.3 Summary of Most Important Sleepiness Prediction Results

Table 4.6 presents a summary of the most important sleepiness prediction results in terms of average MSE across the test drivers in a LOOCV test. Note that both LSTM models perform worse than the baseline model that predicts a constant ORS level, 2, on the scale from 0 - 4. The MSE from the handcrafted eye blink features methods are lower than the constant median prediction.

Table 4.6: Summary of the most important sleepiness prediction results in terms of average MSE across the test drivers in a LOOCV test.

Method	Average MSE
Baseline Model Constant Prediction, ORS = 2	1.72
Baseline Model Random (Uniform) Prediction	3.70
Handcrafted Eye Blink Features - Method B	1.21
Handcrafted Eye Blink Features - Method C	1.32
Handcrafted Eye Blink Features - Method D	1.25
Temporal Features - LSTM 30 s	4.00
Temporal Features - LSTM 120 s	3.54

5

Discussion

This chapter presents the discussion based on the eye state and sleepiness prediction results. The inter-annotator agreement discussion is presented in Section 5.1.1, Eye State Classification Discussion. Furthermore, Section 5.2, Sleepiness Prediction Discussion, is split into two parts: sleepiness from *handcrafted eye blink features* and sleepiness from *temporal features*.

5.1 Eye State Classification Discussion

This section starts with the inter-annotator agreement results presenting how the annotators differ in their annotation technique. Next, the optimization study discussion is presented. Finally, the eye state classification results with decreasing percentages of the data are discussed to learn how much annotated data is required.

5.1.1 Inter-Annotator Agreement Discussion

From the inter-annotator agreement in Section 4.1.1, it is clear that Casper and Anton annotate slightly different, especially on the partially closed eye states. Since the inter-annotated data is not balanced, no general conclusions can be drawn from the metrics. Nevertheless, it is important to acknowledge that manually annotating the eye state is open for different interpretations of the annotation scheme by different individuals. More robust and clear annotation schemes are required to achieve more consistent annotations.

Additionally, the performance of the eye state classification model is highly dependent on the annotations. If the model does not achieve 100 % accuracy, a significant part of the errors might be caused by individual variations when performing the annotations.

5.1.2 Optimization Study Discussion

Based on the argument in the previous discussion section, Section 5.1.1, the best eye state classification model might just be overfitting to the annotations. The best model got an average weighted F1-score of 95.94 %. In contrast, the simplified model got an average weighted F1-score of 94.96 %. Since the decrease in performance is small, the simpler model was selected over the best but more complex model. It is

important to keep in mind that this performance is the average across all drivers from the LOOCV procedure optimized to maximize the average weighted F1-score. A new eye state classification model is fitted to the data for each cross-validation split. After having selected the hyperparameters, the features, and the moving window size $W = 3$, one must keep in mind that the test performance of the eye state classification model is unknown. There is a compromise between utilizing the available data for training the model and testing the model on the data. As a trade-off, the final model was selected based on the hyperparameters found in the `Optuna` optimization study and then trained on all data. To sanity-check its performance, a train/test-split of proportions 7:1 across all drivers was done which is presented in Table 4.3. The performance increased slightly as expected since the test driver is now included in the training set. By knowing this, it is reasonable to assume that fitting the model with all available data results in the best possible model overall.

Furthermore, it is also important to study the misclassifications in the confusion matrices in Figure 4.5. There are very few misclassifications of 2s as 0s (0.48 %) and 0s as 2s (0.35 %). This is important because it can be difficult, even for a human annotator, to determine the difference between partially closed (1) and closed (2) or between open (0) and partially closed (1). Especially if the eyes are on the borderline between the two states. Moreover, it is recommended to further study the model systematically in real-time across different drivers to get a better sense of its performance and at what occasions it misclassifies the eye states. For example, sometimes, the model might incorrectly classify two consecutive eye states as non-open. However, the blink rules described in Table 2.2 requires a blink to have at least three consecutive blink rules to be considered as a valid blink. Therefore, the model is robust against single occurrences of false instances of partially closed (1) and closed (2) eye states. On the other hand, the model is not robust against false instances of open eye state during a blink. For example, for the eye state series 01122**0**221100, where **0** is a false instance of open eye state, this would be considered as two separate blinks by the model.

The project aimed to provide an answer to the question "*Is it possible to determine the state of the driver's eyes using an IR video camera facing the driver?*". By summarizing the discussion above, it is possible up to an average weighted F1-score of about 95 %. However, the performance depends heavily on the dataset and the inter-annotator agreement. Moreover, some errors are more significant than others. For example, classifying open eyes as closed is worse than classifying open eyes as partially closed. Although the loss function used does not consider this, the results show that the model seemed to handle these errors well. Furthermore, suppose the eye state model is used to classify blinks. In that case, it is possible to make it robust against single incorrect classifications with rules like: a blink must contain at least three consecutive frames of non-open eye state.

Another question in Section 1.5, Specification of Issue Under Investigation, is to research the eye state classification model's generalization ability and determine the amount of data required. Figure 4.6 and Figure 4.7 are helpful for this discussion. In Figure 4.6, the within-subject classifiers (trained and tested on the same driver) are

slightly better on average than the model from the LOOCV optimization (train on all except one driver which is the test subject) as expected. However, the difference is quite small. Hence, one could consider the model to have a good generalization ability.

On the other hand, these LOOCV F1-scores do not reflect the actual test scores since the model is hyperparameter tuned in the LOOCV procedure. To get a fair estimate of its test performance, one could do LOOCV on all except a few drivers and test the unbiased performance on these drivers. However, the test performance would be heavily influenced by the choice of drivers in the test set. A possible reason why the generalization ability is good could be that the features from OpenFace 2.0 used to classify the eye states (blink intensity, gaze direction, and the facial landmarks used to calculate EAR) are person normalized. OpenFace 2.0 is already trained on a wide variety of faces and is a generalized system.

Regarding the data required, Figure 4.7 shows that the performance drops when less data is used. It is not easy to judge what performance is enough. However, the performance drop from only using 40 % of the data from each driver is small. The driver with the least amount of annotated data is HPG489. Assuming that 40 % of the data is enough, this would require ∼70 moving windows ending with a closed eye state (2), ∼189 moving windows ending with a partially closed eye state (1), and ∼ 2260 moving windows ending with an open eye state (0). These calculations are based on the number of annotated images and the eye state percentages of 0s, 1s, and 2s in Table 3.1. As an example, 6300 images $\cdot$ 0.40 $\cdot$ 0.028 moving windows with closed eyes (2) / images $\approx$ 70.56 moving windows with closed eyes (2). Nevertheless, it is important to note that these moving windows should be from as many different blinks as possible. If the driver microsleeps for 100 frames and all moving windows are collected from this microsleep, the results will probably not be satisfactory.

To further improve and validate the eye state classification model, it could be helpful to add more drivers to the dataset. Moreover, it could be helpful to benchmark the performance against another dataset, for example, the ability to detect blinks.

5.2 Sleepiness Prediction Discussion

After evaluating the results, it seems as the dataset is difficult to handle regardless of method of choice. Multiple issues have been encountered. First of all, the ORS level is only updated every five minutes. When evaluating sequences shorter than five minutes, one must assume that the actual ORS level is constant within the five-minute sequences. Moreover, obtaining an unbiased and general estimation of performance (ability to classify sleepiness of an arbitrary/average person) of a particular model is something we claim to be impossible for a sample of eight drivers for two main reasons:

I. A sample size of eight persons is unlikely to approximate the general population well.

II. Any fitted model can at most be trained and tuned with seven drivers and be tested on the remaining one if a fully unbiased score should be obtained. Depending on which driver is the test driver, performance will be influenced by the personal characteristics of that driver.

Any results obtained can only be seen as estimations of performance for this particular group of eight drivers, which one could hope might indicate performance in the general case. Although, it is not possible to say anything for sure regarding a model's performance for a general person without adding more drivers to the dataset.

5.2.1 Sleepiness from Handcrafted Eye Blink Features Discussion

In the Section 1.5, Specific Research Questions, the main research question to answer is: "*Can a machine learning system learn to classify the driver's sleepiness level from videos using an IR video camera facing the driver?*". To answer, let us consider that the sleepiness level annotations are subjective. Moreover, let us assume that a tolerance of ± 1 sleepiness levels is accepted due to the subjectivity of the labels. Using the MSE metric, a system that is able to achieve an average MSE ≤ 1 could therefore be considered a successful system. Although, the rate of predictions falling outside the tolerance boundary would also require analysis, for example, if one wishes to minimize false positives of sleepy behavior.

From studying the summary of the sleepiness prediction results in Table 4.6 or the generalization results in Figure 4.16, it is clear that the average MSE is above 1 when using 1 - 7 training drivers. However, by studying Figure 4.10 presenting the test results of Method C, observe that the variance between different drivers is large. For drivers HPG485, HPG487, HPG489, and HPG493, the MSE is less than 1. Because of the limited data set, we cannot confidently say anything from this result. Nevertheless, the results indicate that it might be possible to learn to classify sleepiness from Handcrafted Eye Blink Features on some drivers. On the other hand, let us study the confusion matrices per driver in Figure 4.14. These results are overestimated validation results from Method D since the model is tuned on test data. Still, the confusion matrix from, for example, driver HPG494 (interestingly, the only woman in the dataset) does not indicate that the model has learned anything useful except predicting the median ORS level.

Moreover, assuming an ensemble of eight Random Forest classifiers can be used, an approximate MSE of 1.21 ORS levels was obtained. This score corresponds to Method B, and is calculated based on the average performance achieved by the individual classifiers in such an ensemble. However, this conclusion disregards the fact that consensus among the eight classifiers must somehow be reached, and the average score does not consider this. For example, one could use the majority vote, but as there are eight classifiers there might not be a majority vote for any class. Adding an additional driver (and additional classifier to the ensemble) is probably the best way of resolving this. Otherwise, one could also be removed. Nonetheless, it is unclear whether the average score reflects the real performance that would be

achieved by an ensemble classifier for new drivers, so caution should be taken when considering this method.

Another research question to answer from Section 1.5, Specific Research QUestions, is: "*What is a suitable length (in seconds) of a sequence of images required to classify the driver's level of sleepiness?*". From Method A, the eight best performing Random Forest classifier hyperparameter setups tuned on all except one driver are presented in Table 4.4. According to these results, the hyperparameter setup that achieved the highest performance used a sequence duration of 120 seconds. All other hyperparameter setups used a sequence duration of 180 seconds. Similar observations can be seen in Figure 4.12 from Method D. By studying the sequence duration in the slice plot, it is clear that most trials with low average MSE use a sequence duration of 180 seconds. 120 seconds seems to be the second-best choice, followed by 240 seconds, 60 seconds, and finally 30 seconds.

It is expected that longer sequences lead to better results using *handcrafted eye blink features* since these features are more robust during longer sequences. In contrast, the 240-second sequences seem to fall behind 180- and 120-second sequences in average model performances. A possible explanation for this could be the large percentage of repeated data. In Table 3.6, one can see that by using 240 second sequences, 99.17 % of the data is repeated between two consecutive training instances.

For shorter sequences to work better, we hypothesize that the ORS update interval must be more frequent than five minutes. There is no guarantee that the driver is experiencing the specified sleepiness level at any given part within the complete five-minute sequence. However, the fact that the eye blink features used in this approach become less robust with shorter sequence durations also points out that an entirely different method might be preferable if one wishes to use very short sequences.

Another important research question from Section 1.5, Specific Research Questions is "*How well does the sleepiness prediction models trained on video data generalize to other persons?*". Figure 4.16 illustrates the generalization ability of a Random Forest classifier with tuned hyperparameters. The performance increases with more drivers in the training set. The graph also indicates that a lower MSE could be obtained by adding more drivers to the dataset. Although this assumption might be correct, note that the evaluated model's hyperparameters are tuned based on having six drivers in the training set and one driver in the test set, see Method A. Hence, the model might be biased to work better than expected when six drivers are in the training set. However, it is clear that using seven drivers in the training set still results in a better performance.

Unfortunately, even with these results, it is difficult to answer the research question "*What is the estimated amount and variation of data required in order to build generalized versions of the sleepiness prediction models?*" from Section 1.5, Specific Research Questions. Although the trend indicates that adding more drivers results in better performance, it is impossible to extrapolate this result. Therefore, the unfortunate conclusion is that more data is required to determine the estimated

amount and variation of data required for a generalized model.

5.2.2 Sleepiness From Temporal Features Discussion

As presented in Section 4.2.2, the LSTM model fails the sleepiness prediction task when presented with new unseen drivers. However, it appears to be learning to predict the sleepiness level of previously seen drivers, as the models seems to be overfitting towards the training and validation set drivers. The overfitting indicates that a LSTM-based model, given the time-series features suggested in this report (see Section 3.4.3), might be able to learn to classify sleepiness on unseen drivers. However, without a large amount of computational resources capable of evaluating many different model setups, it seems unlikely that such a combination could be found. Especially since the experiment also needs to be repeated as specified in Section 3.4.1 to obtain a fair, unbiased estimate. One could also try re-introducing methods such as dropout regularization to the network to prevent overfitting. However, this will further increase the hyperparameter search space as suitable values for dropout probabilities are unknown.

Previously, it was mentioned that LSTMs in practice often fail to retain memory over sequences larger than $\tilde{1}00$ time-steps and that the shortest sequences evaluated in this project are compromised of 900 discrete time steps. Perhaps, this is another problem that results in the sub-par test performance observed; maybe the sequences are too long for the LSTM to handle. However, the poor results cannot be fully explained this way. The model managed to overfit on the previously seen drivers in the validation set, which implies that memory might be contained over the sequence.

A potential solution regarding this uncertainty is downsampling longer time series into shorter ones. For example, the 900 steps long sequence, with one value for each image and feature, could be downsampled into a time series of 100 discrete values per feature by averaging nine sequential values together. However, such an approach could also cause aliasing and loss of important information.

Nevertheless, assuming that accurate target labels could be provided for shorter sequences, it would be highly beneficial to find a model with lower sequence duration requirements for classifying sleepiness. The handcrafted eye blink features presented in Section 3.4.2 are notably non-invariant to sequence duration changes. The best models found using these features require sequences of 180 seconds, meaning the start-up time is three minutes. It can be speculated that the raw time-series data from OpenFace 2.0 contains more sleepiness information for shorter sequence durations compared to the handcrafted eye blink features. Thus, a model using the raw time-series data as input could allow for shorter sequence durations and start-up times.

We found no evidence that using longer sequence durations for the LSTM would improve performance compared to using 30-second sequences. However, many unknown variables might affect these results, such as the impact on target label distributions when reducing sequence durations. In order to obtain conclusive results

with eliminated bias when it comes to analyzing the sequence duration impact, the frequency of ORS level updates needs to be increased while still providing a high-quality assessment of the actual sleepiness level of the driver.

Another point in favor of the *sleepiness from temporal features approach* is that an eye state model such as the one presented in Section 3.3 might not be strictly necessary. However, the results from the *sleepiness from handcrafted eye blink features approach* indicate that the eye state contains meaningful information about the sleepiness level. Therefore, it might be of interest to include the predicted eye state vector as an input to the LSTM model.

There are many different variants to the *sleepiness from temporal features approach* that could be explored, given enough computing resources. For example, it could be of interest to switch the LSTM RNN for a Temporal Convolutional Network (TCN) architecture, which have been shown to outperform LSTM models in many tasks. One could also imagine an end-to-end approach utilizing a deep 3D CNN for classifying sleepiness directly on the images, rather than a pipeline extracting spatial input features. However, this would certainly require more data and computational resources.

5.2.3 Feature Importance Discussion

In Section 1.5, Specific Research Questions, two research questions answer are: "*How can the information about the state of the driver's eyes be used to determine the driver's sleepiness level?*" and "*What are the most important visual facial movement actions when it comes to detecting sleepiness using a machine learning model, in regards to the dataset at hand?*".

Previous research has suggested PERCLOS and features related to the eyes opening and closing phase to be indicative of the sleepiness level, see Section 2.1.1. Hence, it is interesting to observe that our feature importance evaluation results in Section 4.2.1.5 also ranks these features high. The top four most important features are related to the eyes opening and closing duration. The next four most important features are related to PERCLOS.

As mentioned, features based on FACS have previously been successfully used for sleepiness prediction tasks in other settings. Unfortunately, the results from the *sleepiness from temporal features approach* in Section 4.2.2 indicate that these features might not be good predictors to use for classifying sleepiness on unseen drivers in this case. However, these types of features should not be rejected as they cannot be determined for sure. There are many other possible explanations as to why the LSTM model failed to produce any accurate results for unseen drivers.

6

Conclusion

The project's main goal was to construct and evaluate a system for classifying a driver's sleepiness level based on the available IR video dataset. In order to achieve this goal, a sub-goal was to build and evaluate an eye state classification model for determining the driver's eye state (open, partially closed, or closed) in the most recent image captured by the camera. Moreover, it was of interest to estimate the required length of a video sequence for determining the sleepiness level. Additionally, since the dataset was limited to eight drivers only, it was necessary to research how the sleepiness prediction model generalizes to other persons. Relevant to this, estimating the amount and variation of additional data required for an accurate generalized sleepiness prediction system was also of interest.

The constructed Random Forest eye state classifier achieved an average weighted F1-score of 94.96% averaged over a LOOCV study. This performance is partly limited by inter-annotator disagreement, and it is not expected that the score could be considerably improved. However, even though the obtained F1-score is comparable to the performance of a human annotator, the model might classify the wrong label in obvious cases. Nonetheless, the eye state classifier was considered good enough for extracting handcrafted eye blink features for usage in the sleepiness prediction task.

Using the handcrafted eye blink features extracted from 120-second sequences, a single Random Forest model trained on seven drivers seems capable of learning to predict sleepiness on the ORS scale with an average MSE of $\sim$1.32. However, the performance per test driver differs a lot, from an MSE of $\sim$0.25 on HPG489 to $\sim$3.2 on HPG488, and is highly affected by inter-individual variability in sleepiness signs and behavior. Nevertheless, it was found that important eye blink features for classifying sleepiness are mainly related to the eye's opening and closing duration and the duration of closed and partially closed eyes during a sequence.

Assuming an ensemble of eight Random Forest classifiers can be used instead, which are all trained on seven drivers, an approximate MSE of 1.21 ORS targets was obtained based on the average performance achieved by the individual classifiers in the ensemble. However, it is unknown what the actual performance would be, as it is expected to differ depending on the chosen method for reaching consensus among the classifiers.

An LSTM fed with time-series features of facial movements and micro-expressions

obtained from OpenFace 2.0 seemed capable of overfitting and predicting sleepiness on previously seen drivers. However, it failed to learn any generalized patterns as it could not predict the sleepiness of unseen drivers. It could be interesting to replace the LSTM with other deep neural networks such as a TCN or 3D CNN. However, a larger dataset containing more drivers is preferable to avoid the necessity of nested cross-validation when evaluating algorithms with heavy computational requirements for training.

In order to classify ORS with shorter sequences than 120 seconds, which might be beneficial for the LSTM model, it is expected that a higher frequency of sleepiness level updates is required during data collection. Using sequence durations shorter than 300 seconds (the ORS update interval in this dataset) reduces the performance as label quality likely decreases. Therefore, the results obtained in this project are likely biased positively toward the longer sequence durations.

Moreover, estimating the number of training drivers required to build a generalized model is impossible, although the results indicate that more drivers in the training data lead to better sleepiness prediction performance. Therefore, it is also essential that more data is collected using additional drivers. Provided enough drivers are added, this action will also eliminate the need for nested leave-one-out cross-validation, consequently simplifying the model selection process.

Our work is based on previous research on sleepiness prediction using video data, mainly in simulated driving environments. To summarize, the results from this work indicate that sleepiness classification from video data could also be possible in a naturalistic driving environment. Therefore, this sleepiness prediction system's continued development could help Volvo Cars in its safety vision; that no one should be killed or seriously injured in a new Volvo. However, adding more drivers to the dataset is of utmost importance to obtain a functional system.

Bibliography

[1] ISO Central Secretary, "Road vehicles — Measurement and analysis of driver visual behaviour with respect to transport information and control systems," Standard ISO 15007:2020, International Organization for Standardization, Geneva, CH, 2020.

[2] T. Baltrušaitis, A. Zadeh, Y. C. Lim, and L.-P. Morency, "Openface 2.0: Facial behavior analysis toolkit," *IEEE International Conference on Automatic Face and Gesture Recognition*, 2018.

[3] A. Zadeh, T. Baltrušaitis, and L.-P. Morency, "Convolutional experts constrained local model for facial landmark detection," 2017.

[4] TadasBaltrusaitis, "Openface 2.2.0: a facial behavior analysis toolkit - wiki - output format." `https://github.com/TadasBaltrusaitis/OpenFace/wiki/Output-Format`, 2019.

[5] "Facs - facial action coding system." `https://www.cs.cmu.edu/~face/facs.htm`. Accessed: 2021-04-12.

[6] "sklearn.ensemble.randomforestclassifier." `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`. Accessed: 2021-04-14.

[7] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

[8] "Road traffic injuries." `https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries`. Accessed: 2021-05-10.

[9] "Facts + statistics: Drowsy driving." `https://www.iii.org/fact-statistic/facts-statistics-drowsy-driving`. Accessed: 2021-05-10.

[10] J. Owens, T. Dingus, F. Guo, Y. Fang, M. Perez, J. McClafferty, and B. Tefft, "Prevalence of drowsy driving crashes: Estimates from a large-scale naturalistic driving study (research brief)." `https://aaafoundation.org/prevalence-drowsy-driving-crashes-estimates-large-scale-naturalistic-driving-st` 2018.

[11] U. Trutschel, B. Sirois, D. Sommer, M. Golz, and D. Edwards, "Perclos: An alertness measure of the past," pp. 172–179, 10 2017.

[12] E. Vural, M. Cetin, A. Ercil, G. Littlewort, and J. Movellan, "Drowsy driver detection through facial movement analysis," pp. 6–18, 11 2007.

[13] S. K. Lal and A. Craig, "Driver fatigue: electroencephalography and psychological assessment," *Psychophysiology*, vol. 39, no. 3, pp. 313–321, 2002.

[14] M. Golz, D. Sommer, U. Trutschel, B. Sirois, and D. Edwards, "Evaluation of fatigue monitoring technologies," *Somnologie - Schlafforschung und Schlafmedizin*, vol. 14, pp. 187–199, 09 2010.

[15] C. Ahlstrom, C. Fors, A. Anund, and D. Hallvig, "Video-based observer rated sleepiness versus self-reported subjective sleepiness in real road driving," *European Transport Research Review*, vol. 7, no. 4, pp. 1–9, 2015.

[16] A. Shahid, K. Wilkinson, S. Marcu, and C. M. Shapiro, "e," in *STOP, THAT and One Hundred Other Sleep Scales*, pp. 209–210, Springer, 2011.

[17] M. Dreissig, M. H. Baccour, T. Schaeck, and E. Kasneci, "Driver drowsiness classification based on eye blink and head movement features using the k-nn algorithm," 2020.

[18] "Smart eye pro | smart eye pro dx." `http://smarteye.se/wp-content/uploads/2020/01/Smart-Eye-Pro-and-DX-TechSpec.pdf`. Accessed: 2021-04-01.

[19] "Smart eye pro." `https://smarteye.se/research-instruments/se-pro/`. Accessed: 2021-04-01.

[20] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, p. 1499–1503, Oct 2016.

[21] T. Baltrusaitis, P. Robinson, and L. Morency, "Constrained local neural fields for robust facial landmark detection in the wild," in *2013 IEEE International Conference on Computer Vision Workshops*, pp. 354–361, 2013.

[22] T. Baltrušaitis, M. Mahmoud, and P. Robinson, "Cross-dataset learning and person-specific normalisation for automatic action unit detection," in *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, vol. 06, pp. 1–6, 2015.

[23] R. Ekman, *What the face reveals: Basic and applied studies of spontaneous expression using the Facial Action Coding System (FACS)*. Oxford University Press, USA, 1997.

[24] Wikipedia contributors, "Decision tree learning — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity`, 2021. [Online; accessed 6-May-2021].

[25] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[27] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[28] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," 2014.

[29] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. 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," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[31] S. Wager, S. Wang, and P. Liang, "Dropout training as adaptive regularization," 2013.

[32] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018.

[33] M. W. Browne, "Cross-validation methods," *Journal of mathematical psychology*, vol. 44, no. 1, pp. 108–132, 2000.

[34] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," *Advances in neural information processing systems*, pp. 402–408, 2001.

[35] "sklearn.metrics.accuracy." `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score`. Accessed: 2021-04-16.

[36] "sklearn.metrics.precision_score." `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score`. Accessed: 2021-04-16.

[37] "sklearn.metrics.recall_score." `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score`. Accessed: 2021-04-16.

[38] "sklearn.metrics.f1_score." `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`. Accessed: 2021-04-16.

[39] Wikipedia contributors, "Mean squared error — Wikipedia, the free encyclopedia." https://en.wikipedia.org/wiki/Mean_squared_error, 2021. [Online; accessed 16-April-2021].

[40] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *25th annual conference on neural information processing systems (NIPS 2011)*, vol. 24, Neural Information Processing Systems Foundation, 2011.

[41] "Hdf5 for python." https://docs.h5py.org/en/stable/. Accessed: 2021-01-25.

[42] "Introducing json." https://www.json.org/json-en.html. Accessed: 2021-03-25.

[43] Wikipedia contributors, "Linear interpolation — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Linear_interpolation&oldid=1004343292, 2021. [Online; accessed 25-March-2021].

[44] M. Boguslav and K. Cohen, "Inter-annotator agreement and the upper limit on machine performance: Evidence from biomedical natural language processing," *Studies in health technology and informatics*, vol. 245, pp. 298–302, 01 2017.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY