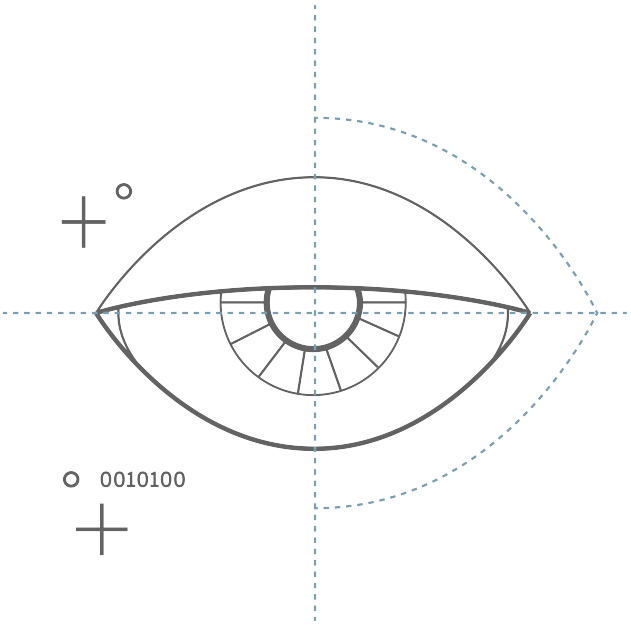

Self-supervised Pre-training for Drowsiness Prediction

Master's thesis in Complex Adaptive Systems

Felicia Ekener
Felicia Ekenstam

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Self-supervised Pre-training for Drowsiness Prediction

FELICIA EKENER
FELICIA EKENSTAM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

Self-supervised pre-training for drowsiness prediction

FELICIA EKENER
FELICIA EKENSTAM

Supervisor: Anton Claesson, Smart Eye
Examiner: Daniel Midtvedt, Department of Physics

Master's Thesis 2023
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Picture of a sleepy eye, connected to drowsiness prediction. Image source: Smart Eye.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Self-supervised Pre-training for Drowsiness Prediction

Felicia Ekener
Felicia Ekenstam
Department of Physics
Chalmers University of Technology

Abstract

Drowsiness experienced while driving poses significant dangers, often leading to severe or even fatal accidents. Consequently, the implementation of technical solutions to mitigate road incidents caused by drowsiness has become essential, with machine learning emerging as a more widespread approach. However, the lack of high-quality data frequently hinders the development of neural networks, also in this domain. Over the past decade, pre-training of neural networks using self-supervised learning techniques has proven to be an effective strategy for alleviating this problem in many cases — which serves as the primary focus of this thesis.

A BERT-like model was pre-trained, using a method inspired by Masked Language Modeling (MLM), on the task of predicting blink features. This was done by considering vectors of blink features, extracted from driving recordings, as token embeddings. The pre-trained model was then fine-tuned on the task of predicting drowsiness. The model's performance was then compared to the performance of a similar model that had not been pre-trained.

The evaluation revealed that the pre-trained model outperformed a set of baseline measures in predicting vectors of blink features for the pre-training task. However, when transferring the knowledge gained from pre-training to the task of drowsiness prediction, it did not achieve better performance than a model without pre-training. One plausible explanation for this observation is the discrepancy between the pre-training task and the drowsiness prediction task, hindering effective knowledge sharing within the model. Several suggestions for future work are proposed, including the optimization of the MLM method and the incorporation of additional pre-training tasks.

Keywords: drowsiness, self-supervised learning, pre-training, fine-tuning, BERT

Acknowledgements

We would like to express our sincere gratitude to our supervisor, Anton Claesson, from Smart Eye, for his invaluable support and guidance throughout the duration of our thesis. Anton's expertise and dedication have played a crucial role in the successful completion of our research. We would also like to extend our heartfelt appreciation to Raimondas for his invaluable contribution to our understanding of the project and the process of writing this report. We would also like to thank our examinator Daniel Midvedt at Chalmers for providing feedback on the project.

Felicia Ekener and Felicia Ekenstam, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations for Transformers
DMS	Driver Monitoring System
DNN	Deep Neural Network
GAN	Generative Adversarial Network
KSS	Karolinska Sleepiness Scale
MAE	Mean Absolute Error
MLM	Masked Language Modeling
MSE	Mean Squared Error
NLP	Natural Language Processing
NSP	Next Sentence Prediction
ORS	Observer-rated sleepiness
SRS	Self-reported sleepiness
SSS	Stanford Sleepiness Scale
VAE	Variational AutoEncoder

Contents

List of Figures

List of Tables

1

Introduction

This chapter presents an introduction to the concept of driver tiredness and its relation to road safety and the motivation for doing this project in relation to EU road regulations. It also covers a section on drowsiness prediction, collection of data, and the importance and difficulty of having high-quality labels. To be able to explain the aim and process of the thesis, the background also includes an introduction to the transformer language model BERT and the methods it utilizes. Then, the chapter also covers the aim, research questions, delimitations, and limitations.

1.1 Background

Tiredness while driving is one of the main causes of road accidents today [1]. With an accurate system for detecting how tired a driver is, proactive measures can be employed before a critical level of tiredness is reached. If such systems were implemented in cars it could improve road safety and potentially prevent fatal accidents from occurring. Other than the probability of preventing fatal accidents, another motivation for such Driver Monitoring Systems (DMS) is the new EU road regulations. They state that DMS is to be mandatory in all vehicles produced on the EU market by 2024, which is believed to lead to at least 140,000 avoided serious injuries by 2038 [2].

The task of estimating the tiredness level of a driver is referred to as drowsiness prediction and can be formulated as a regression problem [3]. To solve this problem, amongst others, Smart Eye is developing DMS software using machine learning algorithms. The drowsiness prediction models are trained using videos collected with IR cameras in real-world driving scenarios. The purpose of the videos is to capture the driver's behavior in different driving scenarios and transform the videos into raw data. By analyzing the data it is possible to relate the behavior to stages of drowsiness, with labels produced according to a selected sleepiness scale [4]. However, collecting data with high-quality drowsiness labels for supervised learning is often difficult, time-consuming, and expensive. Thus, a way to enable training on unlabeled data could help to mitigate this bottleneck.

2018, Devlin, Chang, Lee, *et al.* published a paper about the transformer-based model named BERT, standing for Bidirectional Encoder Representations from Transformers. It demonstrated the capabilities of self-supervised pre-training on large

amounts of unlabeled data [5]. The self-supervised pre-training allows the BERT model to learn general concepts of text data, which is useful and can be transferred to multiple downstream tasks where labels are available. To enable self-supervised learning, BERT makes use of two techniques, named Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The BERT model proved to yield state-of-the-art results on Natural Language Processing (NLP) tasks upon release and together with the paper on Attention by Vaswani, Shazeer, Parmar, *et al.* it has popularized transformer models [6].

Transformer language models, such as BERT, are widely used within the NLP area [5]. During recent years the developments and performance of transformers have grown immensely in many other areas such as vision, and audio, which is a great motivation for trying out the transformer on other domains such as blink sequences [7]. Hence, this thesis will examine the possibilities to develop a model, inspired by the BERT model, which takes time series data of drivers' blinks while driving, collected at Smart Eye, as input instead of text data. The hope is that the model can help to improve the prediction of drowsiness levels, as well as introduce training with a smaller amount of high-quality labels.

1.2 Aim

This project aims to evaluate the impact of self-supervised pre-training regarding drowsiness prediction. A transformer model, inspired by BERT [5], will be pre-trained on sequences of blink embedding vectors using a self-supervised method similar to masked language modeling. The input data will consist of sequences of blink features, derived from driver videos collected by Smart Eye, where a substantial amount of it doesn't have corresponding drowsiness labels. The model's pre-training objective is to perform self-supervised training on the subset of unlabeled data. Later on, supervised drowsiness regression fine-tuning can be performed on the remaining subset of labeled data.

The performance of the fine-tuned model with pre-training will then be compared to the performance of a model that hasn't been pre-trained, to evaluate the impact of the pre-training. Furthermore, both the pre-training and fine-tuning will be done with different methods, e.g. different sizes of training data and different unfreezing strategies, to find an optimized model concerning performance as well as size of training data.

1.3 Research Questions

The purpose of this study can be summarized in the following research questions:

1. To what extent is it possible to predict blink features in a sequence, using self-supervised learning?
2. Does self-supervised pre-training on sequential blink data improve drowsiness prediction?

3. Does adding more data during pre-training improve the performance of drowsiness prediction?

1.4 Delimitations

- The self-supervised learning technique implemented in the model in this thesis is derived from the MLM technique from the BERT model. The other technique used to pre-train BERT, NSP, was not implemented, partly due to lack of time.

1.5 Limitations

- There are a lot of choices and settings related to the model that affect its performance, e.g. hyperparameter settings, model architecture, and data processing. Several different methods were applied to optimize the model. However, the methods are both time-consuming and require high GPU utilization. An ideal model is therefore difficult to find, which might influence the performance of the model.
- The labels corresponding to the drowsiness data used in the project are self-reported, i.e. more or less subjective. Even though the drivers that performed the data collection were trained to interpret the used drowsiness scale as equally as possible, the risk for diverse perceptions of drowsiness can't be fully eliminated. Therefore it is likely that the data contains quite a bit of noisy labels due to the subjective nature of the scale.
- When working with a company in a competitive line of business there can be a lot of elements that are limited due to secrecy. Therefore there are some factors that can't be described in full detail, e.g. how the data has been processed, and how blink feature vectors and Smart Eye's internal metrics are calculated. This is especially relevant for describing the data since most parts of the data are confidential.

2

Theory

This chapter contains the theory for this thesis focusing on self-supervised pre-training. Section 2.1 presents the definition of drowsiness, different sleepiness measurement scales, and how the data used in this have been collected. Sections 2.2 - 2.4 provide a brief introduction to artificial neural networks (ANNs), and the difference between the learning approaches supervised, unsupervised and self-supervised learning. Further on, sections 2.5 - 2.6 give insight into transformers, token embeddings, attention, and BERT. Lastly, section 2.8 describes the task of optimizing the hyperparameters in a machine learning model, meanwhile, section 2.7 presents the different metrics used in this thesis to measure model performance.

2.1 Drowsiness

Drowsiness, also known as tiredness, is defined as a tired state, between sleeping and being awake, that is related to the circadian rhythm [8]. It is mostly connected to nighttime when the human body naturally wants to sleep and slow down its activity and alertness. However, measuring drowsiness is a very complex task, yet a desirable feature in vehicle safety systems in order to avoid accidents.

Measuring drowsiness when driving can be done by observing and analyzing behavioral and physiological signs in the driver. Behavioral signs in driving include slow reaction time, inattention to the environment, and an inability to maintain a fixed speed. Meanwhile, physiological signs include for example frequent yawning, micro-sleeping, and problems with keeping the eyes open. In relation to the last mentioned sign, according to Belz, a possible way of measuring drowsiness is by looking at different aspects of a blink such as blink duration, blink frequency, delay in lid reopening and so on [9].

2.1.1 Collection of Drowsiness-data

As in most classification or prediction tasks, the need for high-quality ground-truth data is necessary for good performance. However, the subjectiveness of drowsiness makes it non-trivial to gather high-quality annotations of drowsiness data. Two of the most common approaches to collecting drowsiness data are observer-rated sleepiness (ORS) and self-reported sleepiness (SRS) [10]. The basics of ORS is that an observer gets to estimate the level of sleepiness of a driver by either being in

the car with the driver or by looking at video recordings afterward. In both cases, the observer will look at facial expressions, duration of eyelid closures, and body movements. In SRS, on the other hand, it is the driver itself that reports how they are feeling according to the sleepiness scale.

The data used in this thesis was collected by Smart Eye through multiple different driving sessions, with a large set of drivers who were not employees. The sessions have taken place mainly in real driving sessions in traffic, during varying durations, and at different times of the day. The drivers have been filmed during the sessions, thus, the raw data consists of videos. During the sessions, the drivers reported every five minutes at which level they perceived to be on the sleepiness scale. I.e. the data labels were collected using SRS. After the driving session, the KSS labels were completed by assigning each assessed value for the upcoming five minutes, since those timeslots hadn't been assigned a specific KSS value during the driving session.

2.1.2 Karolinska Sleepiness Scale

When measuring drowsiness, there are multiple different scales and approaches to use [4]. The most commonly used scales in similar projects are the Stanford Sleepiness Scale (SSS) [11] and Karolinska Sleepiness Scale (KSS) [12]. Both scales assess sleepiness at a determined time, unlike other similar scales which evaluate sleepiness over the course of a day. The measurement used for the level of drowsiness in this project is the Karolinska Sleepiness Scale (KSS), which is presented in table 2.1. It is a subjective alertness scale where the driver itself gets to record their experienced drowsiness level at a certain interval, during a long time of driving.

Table 2.1: KSS scale. The different sleepiness levels of the Karolinska Sleepiness Scale (KSS) [12].

KSS point	Description
1	Extremely alert
2	Very alert
3	Alert
4	Rather alert
5	Neither alert or 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, fighting sleep

In regard to Table 2.1, the most critical cases in drowsiness prediction would be when the real label would be a 7-9, meanwhile, the model predicts a 1-3. This would imply that the driver is in a dangerous situation of possibly falling asleep while driving, but the system doesn't catch that and therefore doesn't give a warning.

2.2 Artificial Neural Network

Historically, neural networks refer to networks of neurons in the brain. Nowadays, neural networks in the context of machine learning, i.e. artificial neural networks (ANNs), have become a widely-known concept. ANNs are inspired by the neural network in the brain, with the fundamental principle to be able to learn from data by changing connections between the neurons in the network [13]. In practice, ANN is a mathematical or computational model and is a highly idealized version of a biological neural network.

The characteristics of an ANN provide it with the ability to perform rather advanced information-processing tasks, e.g. recognize structures and trends in data. For example, given a training set of data and corresponding labels, or target values, an ANN can be trained to classify the data by adjusting the connection strengths between its neurons. When the network is trained, it can generalize the results to other data sets, given that the new data is similar to the training set. ANNs can also train on unlabeled large datasets, often of high dimensionality. In these cases, the learning algorithms organize the data in different ways, so that e.g. clusters och structures of the data can be found. More detailed explanations about different learning techniques are presented in the following sections.

2.3 Supervised and Unsupervised Learning

Supervised and unsupervised learning represent two approaches for training neural networks [13]. Supervised learning requires a data set that only consists of labeled data for a specific target problem. From these labels, a predictive model is trained to estimate their targets. The model often includes a representation extractor function and a classifier or regression function, which are trained by minimizing a loss function (see section 2.7). On the contrary, unsupervised learning models are trained only on unlabeled data by building generative models or density estimators (e.g. Variational AutoEncoders, VAEs, and Generative Adversarial Networks, GANs), or compact latent representations (e.g. autoencoders, clustering).

Two of the most common supervised machine learning techniques are regression and classification, which, in general, are suitable for different types of problems. The main distinction between them is that regression aims to determine continuous values, whereas classification estimates discrete values, i.e. classes.

The general method in regression is to find a model or function which maps input variables to a continuous output value [14]. This makes it suitable for problems such as e.g. probabilities, time, or prices. Furthermore, since it predicts a quantity, its performance is measured by a loss function, i.e. a function that calculates a loss on the prediction based on how far away it was from the label. However, this is not only the case for regression, since loss functions are a widely used element in measuring performance or optimizing models. For more on this, see section 2.7.

Classification on the other hand aims to find a model or function which can divide

the input data into the correct, discrete, class. A classification problem can be solved by initially using a regression algorithm e.g. by predicting the probabilities for the input data to belong to each class. However, the main characteristic, that the output is a discrete value, is sustained by adding a final output element that transforms the highest probability into their respective class label, thus, it's still a classification task.

2.4 Self-supervised Learning

The annotation of data is often costly and time-consuming to collect, resulting in it being a bottleneck for the application of machine learning. As a consequence, there has been a lot of research focusing on a new field within machine learning during the last ten years, called self-supervised learning [15]. Self-supervised learning isn't too different from unsupervised learning, as both techniques rely on unlabeled data. What separates them is that self-supervised learning has feedback loops, which unsupervised training doesn't. To enable the feedback without any labels available, self-supervised learning uses techniques to auto-generate pseudo-labels from the input data, which are then used for training. A visualization of the different learning types is shown in Figure 2.1.

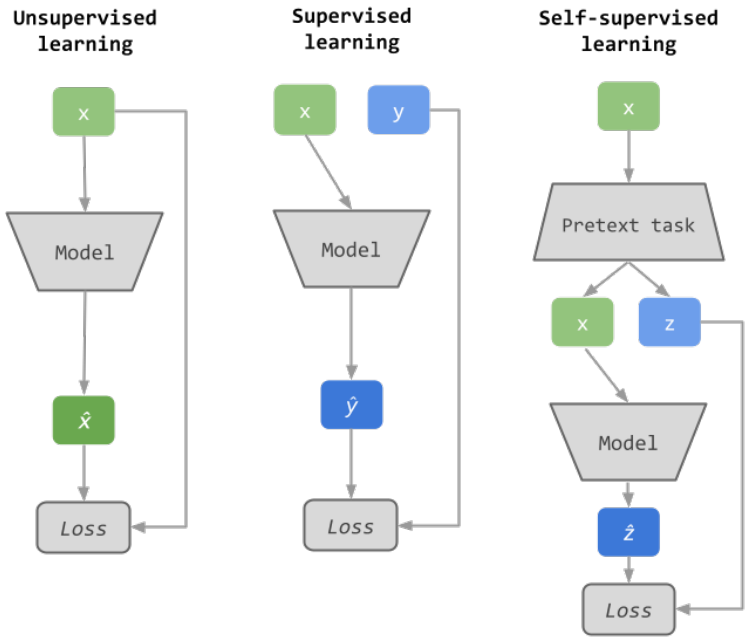

Figure 2.1: Unsupervised-, supervised- and self-supervised learning. Visualization of different types of learning in models, where x is a dataset which is used as input, y is existing labels linked to x, and z is pseudo-labels produced by the pretext task.

As described, self-supervised learning can be an efficient method in cases with unlabeled data. Furthermore, it's also useful in cases when there's a large amount of

data available, but only a share of it is labeled. In these cases, self-supervised learning can contribute to general-purpose training on the unlabeled data, which can be transferred to other downstream tasks when labeled data is available. However, the choice of pretext task can be crucial to achieving good results with this method [16].

The general process of using self-supervised training and transferring it to a downstream task can be described in four steps.

1. The data set created for training should consist of a small set of labeled data and a larger set of unlabeled data.

2. Depending on the chosen pretext task, the model generates a data set with pseudo-labels which is repeated at the start of each training epoch.

3. The pretext model is trained to optimize the desired objective on the pseudo-data set, i.e. it estimates the model's parameters without any annotated data.

4. Finally, the pre-trained weights from the pretext task are fine-tuned by being trained with supervised learning on the labeled data.

When transferring learning, it's common to use different techniques regarding how susceptible the transferred network should be during fine-tuning, i.e. whether or not the layers in the pre-trained model should be frozen or not. To freeze a layer means that the weights are fixed during training and therefore not updated through the backpropagation [17]. Thus, when transferring learning, it's common to initially freeze all layers in the pre-trained network, and then gradually unfreeze a number of layers throughout the training process. The first layers of a neural network tend to learn general features of the dataset, meanwhile, the top layers learn more task-specific features [18]. It is therefore more crucial to gradually unfreeze the top layers for them to adjust to the new task to be able to reach better performance. This technique involves making choices regarding for example how many times to gradually unfreeze layers, at which epochs to do it, and also how many layers to unfreeze at each time. A potential drawback of unfreezing the whole model is a higher risk of overfitting and forgetting already learned information, therefore it is important to evaluate when and where to start unfreezing.

2.5 Transformers

A transformer is a neural network model originally designed for Natural Language Processing (NLP) tasks [6]. However, it has gained significant popularity and achieved state-of-the-art performance in various domains such as vision and audio processing. The progress in transformer models has led to the development of a diverse range of architectures while sharing common underlying methods and mechanisms. One fundamental mechanism found across different transformer architectures is the attention mechanism [6], which will be described in further detail in a later section, as well as other vital elements of a transformer.

2.5.1 Tokenization

Natural language processing tasks often handle natural language in the form of either text or speech [19]. To be able to work with raw language data it is first necessary to transform it into usable data formats. An example of this is tokenization, which is used to split sentences into smaller tokens (words, subwords, or characters) that can be assigned a specific meaning. The task of tokenization will help the model understand the tokens on their own, as well as the larger context of the sentence since the tokenization is often done with respect to a specific vocabulary of tokens.

2.5.2 Embeddings

Embedding is an essential technique used in transformer models for NLP tasks such as translation, classification, and text generation [20]. Embedding is the process of representing the tokens in a sequence as high-dimensional vectors, where each dimension corresponds to a specific feature of the token. As mentioned, in NLP tasks the text data is often presented as sequences of tokens that transform into embedding vectors. However, in some cases, it can be useful to perform feature upsampling to increase the dimensionality of the input data. Feature upsampling is used to improve the quality of the embedding and enable the model to capture more complex relationships between the input tokens. When performing feature upsampling it is required to set a suitable embedding size. Although, the choice of upsampling should be based on the specific data and task at hand. It may not always be necessary or beneficial, and sometimes it can be more beneficial to reduce the feature dimensions. It all depends on the task-specific data and desired end goal.

In the task of feature upsampling, there are multiple different ways of performing it depending on the desired task. One approach is to expand the features to a higher dimension by applying a linear transformation to the incoming data. This increases the input feature dimensionality but without taking into consideration the relationships between the features and input tokens. Another approach is attention-based upsampling, which uses the self-attention mechanism (see section 2.5.4 to represent the relationships between the input tokens as well as increase their dimensionality [21]. Applying self-attention allows the model to attend to different parts of the sequence to capture more complex relations between tokens. Kundu, Mostafa, Sridhar, *et al.* used a similar approach of attention-based image upsampling on super-resolution tasks, to compare the performance to more traditional upsampling methods using for example convolutions. Hence, the process of upsampling on numerical, three-dimensional data is not identical to image upsampling due to different data dimensions, but a similar approach is still feasible.

2.5.3 Positional Embeddings

After the input embeddings are in order, the next step is usually to add positional embeddings. The purpose of positional embeddings is to give the model information on the relative positions of tokens in a sequence to enable the processing of sequential

data and capture long-range dependencies [6]. This means that without positional embeddings, the model would only rely on the order of tokens in the input sequence when creating connections between words. Hence, by adding positional embeddings, element-wise to the input embeddings, it allows the model to distinguish between tokens based on their positions, leading to it capturing sequential patterns and dependencies in the data.

The positional embeddings consist of a set of vectors representing the positions of tokens in a sequence, which are learned during training. These vectors can be calculated in various ways, e.g. by using an embedding layer. The input to the layer is not a token, instead, it is the position of the token within the sequence. The positional embedding is made using a matrix with the shape of the maximal position, i.e. the max length of a sequence, times the embedding size to capture enough information within a sequence to build dependencies.

2.5.4 Attention

As earlier mentioned, transformers make use of something called the attention mechanism [6]. The role of attention is to weigh the importance of each part in a sequence, in relation to the other parts. I.e. it will look at an input sequence and learn to decide which other parts of the sequence are important at each step. For example in NLP tasks, the attention could for each word in a sentence look at the other words, and determine how relevant they are for the current word's meaning. The specific attention for each word is assigned by a weight or a score, representing their importance to the other words.

The main elements in an attention mechanism are called query (Q), keys (K), and values (V), and the attention is computed as a dot product of them. Furthermore, the dot product can be scaled, which describes a specific type of attention mechanism called Scaled dot-product attention, which is calculated as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

where d_k is the dimension of K, which the dot product is scaled down by.

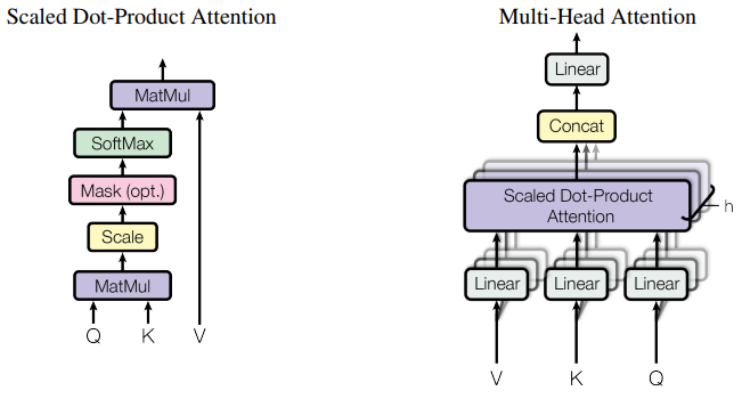

Figure 2.2: Scaled dot-product and multi-head attention. The architecture of the Scaled dot-product attention and multi-head attention, is given in [6].

Scaled dot-product attention is one of the most commonly used attention mechanisms in NLP tasks. One of the other most common is Multi-head self-attention, which essentially is a stack of multiple identical attention mechanisms, called *heads*. The data is split across the heads, which enables them to process it independently and in parallel. Then, the full multi-head attention is calculated by concatenating the attention from each head. This approach allows the network to learn information about subsets of the data, and thus a richer representation of it. In summary, multi-head attention can be described as

$$\begin{aligned} \text{Multi-Head Attention}(Q, K, V) &= \text{Concat}\left(head_1, head_2, ..., head_h\right) W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.2}$$

and W^O, W_i^Q, W_i^K and W_i^V are parameter matrices. The architecture of both of these two types of attention can also be seen in Figure 2.2.

2.5.5 Encoder and Decoder

The encoder-decoder structure is a form of neural network architecture used for tasks with a sequence of data as both input and output [6]. Examples of such tasks include machine translation, question answering, and text summarization. In this architecture, the encoder component converts the input data into a fixed-length internal representation that contains sufficient information for the decoder to process and generate the desired output. The decoder, in turn, takes the encoded data and decodes it into the required output format. For example in NLP tasks, the encoder-decoder structure takes words in a text as input, encodes them to e.g. a two-dimensional vector, and decodes them into e.g. another language.

The encoder in the encoder-decoder structure consists of two sub-layers: a multi-head self-attention mechanism and a position-wise feed-forward network (FFN). The FFN, also known as an expand-and-contract network, consists of two fully connected

layers. The first layer, i.e. the hidden layer, expands the input from the embedding size to a hidden size, generally four times the embedding size. An activation function, often Rectified Linear Unit (ReLU), follows the first layer to speed up training by selectively activating neurons based on their input values. ReLU outputs 0 if the input is negative and keeps the input value otherwise, allowing only the neurons corresponding to positive values to be activated during gradient calculations. The second layer is a fully connected layer that contracts the input to the embedding size. The purpose of the FFN is to add more element-wise non-linearity transformations in the input before passing it to another attention layer or the decoder. Both sub-layers utilize a residual connection, which normalizes the input and adds it to the output of the sub-layer.

In contrast, the decoder has an additional sub-layer compared to the encoder. This third sub-layer is a type of multi-head attention called encoder-decoder attention, which also has a residual connection around it followed by normalization. This attention mechanism allows the decoder to generate output tokens by weighing the relevant elements from the encoder's output. However, the difference is that it works by masking the future positions so that it only can attend attention to the tokens already generated by the decoder in earlier steps. The purpose of it is to prevent the model of cheating by looking forward in the sequence and thereby seeing the other words, which have not yet been generated or predicted by the model.

2.5.6 Transformer Architecture

From the information in the sections above, the entire architecture of the transformer can now be explained as a whole. As seen in Figure 2.3, the input to the encoder first goes through an embedding layer, to transform it to the desired feature dimension as meantioned in section 2.5.2. Then, the input gets added with a positional encoding before being passed into the encoder. The encoder is constructed by N identical, stacked layers containing the two sub-layers, multi-head self-attention, and FFN. Next, the output of the encoder is passed on to the decoder which is also a stack of N identical layers, with the three sub-layers described above.

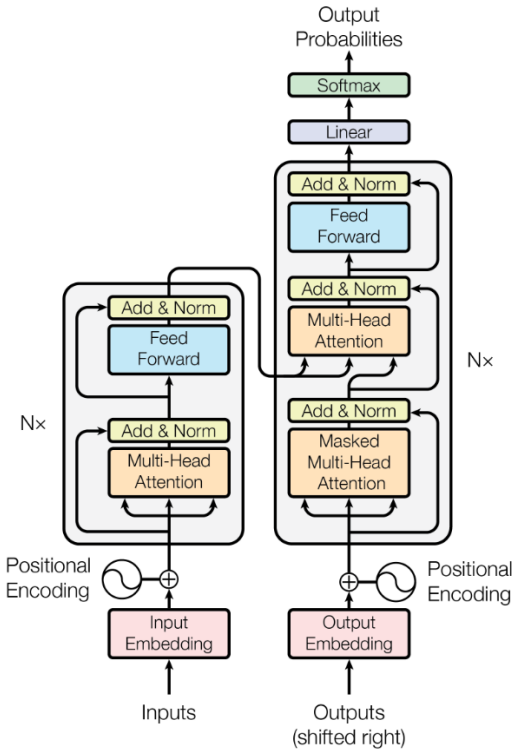

Figure 2.3: Transformer model. The architecture of the transformer, containing an encoder and a decoder, is given in [6].

The output of the decoder is passed to an output head consisting of a linear layer followed by a softmax activation function to convert the decoder output to predicted next-token probabilities.

2.6 BERT

Historically, language models have only been able to read input text unidirectionally, e.g. right-to-left or left-to-right. 2018 Devlin, Chang, Lee, *et al.* published a new transformer-based model called BERT, which stands for Bidirectional Encoder Representations for Transformers. One of the main contributions of the BERT model is a demonstration of how the input text is read bidirectionally instead of unidirectionally thanks to one of its pre-training techniques.

BERT's architecture can be divided into two parts, pre-training, and fine-tuning, where the first part is trained with self-supervised learning and the second with supervised learning. During pre-training, two different pre-training tasks are implemented to enable training on unlabeled data, masked language modeling (MLM) and next sentence prediction (NSP), which contribute token-level learning and learning about the relationship between sentences respectively. This mechanism behind them will be described further in the following sections.

When the pre-training has been done with the pre-training tasks as objectives, the

final layers are replaced with a new task-specific architecture for other downstream language tasks. In other words, during the fine-tuning part, the BERT model is initialized with the pre-training parameters and then fine-tuned using labeled data from the downstream task.

Some examples of fine-tuning tasks BERT was tested on are diverse natural language understanding tasks, predicting answers in a passage of text related to a given question, and choosing the most plausible continuation to a sentence among four choices. In all tests, BERT showed better performance than the other models it was compared with.

2.6.1 Masked Language Modeling

As mentioned before, one of the self-supervised training techniques used in the pre-training part of BERT is MLM [5]. The method is to randomly choose words in sentences that are masked during training, and the objective is to predict the words that are hidden behind the masks. In the BERT model, 15% of the data is chosen for MLM. However, replacing all chosen words with a mask token creates a mismatch between pre-training and fine-tuning, since the mask token doesn't appear during fine-tuning. This obstacle is mitigated by only replacing 80% of the randomly chosen words with the actual mask token. Then, 10% of the randomly chosen words are replaced with the true word and finally, the last remaining 10% of the chosen words are replaced by a random word.

One of the main benefits of MLM is that it allows the model to train bidirectionally instead of unidirectionally. Unidirectional and bidirectional language models differ in how they read the context, i.e. other words in a sentence. For example, a unidirectional model will predict a word based on the text that precedes, left-to-right, or follows it, right-to-left, in the sentence. Bidirectional language models, like BERT, on the other hand, can read both the preceding and following text in the sentence, which evidently contributes more information to the prediction. In practice, it's not trivial to implement bidirectional models by simply joining left-to-right and right-to-left reading, since it would "reveal" the word to be predicted in a multi-layered model. This obstacle is however overcome by how MLM operates. Since the target words are masked, and thus aren't visible as the data is processed through the model, they won't be revealed. MLM is therefore the factor that makes the bidirectional conditioning possible.

2.6.2 Next Sentence Prediction

The other unsupervised technique used during pre-training of the BERT is NSP, which is used to train the understanding of the relationship between two sentences. NSP chooses two sentences, A and B, where B is the actual next sentence that follows A 50% of the time, and the other 50% it is not. Accordingly, the task is a binary prediction of whether B is the next sentence or not. In 2019 Liu, Ott, Goyal, *et al.* presented a new version of the BERT model called RoBERTa which found that removing the NSP objective led to matched or slightly improved downstream

task performance. Even though the RoBERTa made other changes that could have contributed to the improved performance, it could still be a motivation for choosing to not include NSP in this thesis.

2.7 Metrics and KPIs

Two important elements within the field of machine learning are metrics and objective functions, i.e. loss functions. Metrics are used to evaluate the performance of a model after training and testing, meanwhile, loss functions are implemented to optimize the model during training. I.e. loss functions are measures of how well a model can fit a given data set, by calculating the difference between a predicted value and the real input target [23]. However, some of the most commonly used loss functions can also be used as evaluation metrics and vice versa.

The two most common loss functions in regression tasks are Mean Absolute Error (MAE) and Mean Squared Error (MSE) [23]. MAE is the mean absolute value of the difference between the actual target and the estimated target, whilst MSE is the mean squared difference between the target and the estimated value. The equations for them are

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.3}$$

and

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.4}$$

where N is the number of data samples, y_i is the actual target, and $\hat{y}_i$ is the estimated target.

As the data in this project mainly consists of continuous values, regression algorithms are used for most of the objectives, and thus MAE and MSE are used then. The measure for drowsiness is, however, based on a discrete sleepiness scale. To connect the continuous predictions from the regression algorithms to the discrete sleepiness scale, the continuous predictions are in some cases rounded off to integers. By doing this, classification analysis, connected to the sleepiness scale, is enabled. This does however also give rise to a need for other metrics and methods to evaluate the results.

A performance measurement that is often used in many contexts is Key Performance Indicator (KPI) [24]. KPIs are used to measure how well a desired end goal is met, rather than calculating an overall performance. At SmartEye, an internal KPI has been developed which measures the drowsiness prediction performance. The continuous KPI value is calculated based on discrete drowsiness classifications, either on a 9-level scale as the original drowsiness scale is ranked or on a 4-level scale where the 9-level predictions are binned in four levels instead. How the KPI is calculated will however not be explained in further detail, due to secrecy.

Another way to analyze the classification results is by confusion matrices. A confusion matrix describes the results by categorizing them into True Positives *(TP)*, False Positives *(FP)*, True Negatives *(TN)*, and False Negatives *(FN)*. The ground-truth values in a data set are *True* and *False*, which are classified as *Positive* or *Negative*. Thus, a confusion matrix contains the number of samples that were correctly and wrongly classified. Furthermore, a confusion matrix can be produced on binary as well as multi-class cases. A visual representation of a binary example is shown in Figure 2.4.

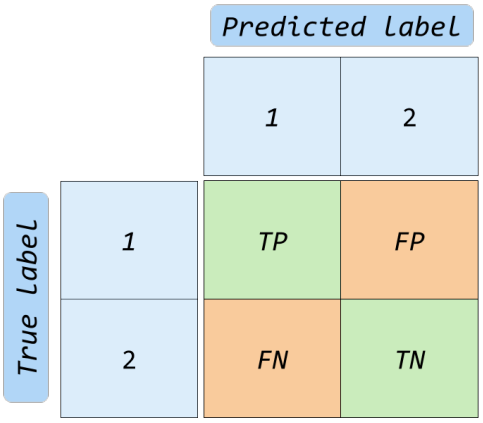

Figure 2.4: Confusion matrix. A plot showing how a confusion matrix is calculated through the rates True Positives, False Positives, True Negatives, and False Negatives.

2.8 Hyperparameter Optimization

To obtain the best results from a machine learning model, it is common to use some sort of hyperparameter optimization algorithm to explore different settings throughout the parameter space. Most machine learning models have approximately around ten to fifty different hyperparameters depending on the type of model and structure [25]. The parameter space consists of the different parameters and a set of intervals or choices for each parameter. The intervals can contain either discrete or continuous values, as well as a set of logical or categorical values, and they can vary rather much depending on the desired task.

The task to find the optimal parameter settings can be very tedious and time-consuming to do manually if there is a big set of parameters. Thus, there are many optimization programs available, used to find appropriate parameters that optimize a task-specific loss function. Optuna, the program that is used in this thesis, is an optimization software framework with Define-by-run programming, i.e. it allows the user to dynamically construct the search space [26]. The basics of it are to run a certain amount of trials with different combinations of parameter values to find the set of parameters contributing to optimal model performance. Optuna lets the user build a parameter search space and define the settings of how the algorithm will explore the parameters to optimize a selected loss function.

After constructing a search space, by choosing the different intervals and values for each hyperparameter, the algorithm starts by choosing a random set of parameters for a certain amount of trials. The random trials are executed to evaluate how the different parameter values affect the model's task-specific performance. But instead of continuing to do a random search throughout the entire search space, the algorithm takes the previous performances into consideration when choosing new setups. If a selected setup of parameters results in poor performance early on the algorithm can terminate the current trial and not waste space and resources on unwanted performances. The method of early terminating trials is a necessary part of optimizing and decreasing search times. Also, to be more certain of the result from a sweep it is recommended to run between 100-1000 trials, to truly investigate the different combinations.

The optimization algorithm in Optuna is mainly the Tree-Structured Parzen Estimator (TPE) algorithm, which is based on independent sampling [26]. The two most common sampling methods are relational sampling and independent sampling, where the first one utilizes the correlations between parameters and the second doesn't. Optuna can handle both independent and relational sampling methods and the choice depends on the task and desired outcome.

3

Methods

The main task of this thesis was to build a self-supervised pre-training model and a supervised fine-tuning model. The concepts of the models were based on the BERT model, described in section 2.6. The pre-training model used Masked Language Modeling (MLM) to enable self-supervised training. Furthermore, both models were designed with a backbone and a head, where the pre-trained backbone was re-implemented in the fine-tuning model. The general architectures of the models are shown in Figure 3.1.

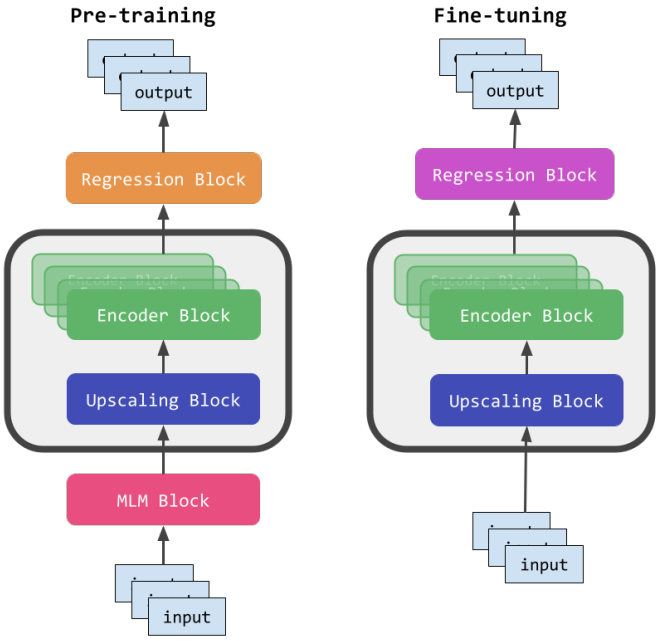

Figure 3.1: Pre-taining and fine-tuning models. The general architecture of the pre-training and fine-tuning models, where the backbone from the pertaining model, containing the upscaling- and encoder blocks, was re-implemented in the finetuning model.

The data underwent a series of data management procedures before it could be used in the models. The primary objectives of these data management methods were to prepare the data in a format suitable for the specific task in this project and to mitigate any potential issues such as data leakage.

After the data had undergone the necessary processing and the models had been

designed, a series of iterative runs were conducted to determine the optimal configurations and ultimately yield outcomes that provide results that could answer the research questions. Further description of all components of the methodology will be presented in the following sections.

3.1 Data Management

The data used in this project consisted of processed data obtained from Smart Eye's driver monitoring software. The models were therefore not fed with raw images or videos, but with numerical data that had been extracted from the videos of the different drivers. Specifically, this thesis focused solely on features related to drivers' eye blinks, with a selection of up to nine relevant features. Due to confidentiality, the specific features can not be described in full detail. However, a majority of them are related to some form of duration, for example, the duration of an entire blink. In addition to the original features, there were a number of additional features that were derived from the original ones, e.g. calibration features, which could be used for standardizing the blink features across all drivers.

To ensure compatibility with the methodology, various data processing techniques were applied. The original data consisted of one individual data frame for each recording, where every row represented a blink and the columns represented the different blink features. These data frames were utilized to create a time series dataset composed of subsequences extracted from each recording. These samples were generated by employing a moving window of a predetermined sequence size. The window created a subsequence, then shifted one row to produce another subsequence, and so forth. Accordingly, the generated subsequences were partially overlapping with one another.

For the models to operate properly, it was necessary to have data frames of uniform shape, specifically with equal subsequence lengths. Consequently, during the construction of the data frames, measures were implemented to ensure that subsequences did not contain rows from different recordings. In cases where sequences did not reach the maximum length, augmentation was performed by appending a PAD token at the end, typically represented by the value 0.

Furthermore, the data had to be split into two more data sets, one for pre-training and one for fine-tuning. It was crucial to prevent any overlap of drivers between them in order to avoid data leakage. Furthermore, the issue of data leakage had to be considered during the split into training, validation, and test sets as well.

The fine-tuning data set was constructed to only include high-quality data, while the pre-training set included any data with blinks, including lower-quality labeled data. High-quality data refers to instances where the drivers responsible for generating the labels have acquired a high level of competence regarding the selected sleepiness scale, resulting in a reduction of subjectivity.

When splitting the fine-tuning data set, a weighted sampling approach was employed to maintain balance with respect to the labels. This means that the data was split

into training, validation, and test sets, ensuring an equal representation of samples from each label category. This strategy aimed to prevent the model from exhibiting a bias toward predicting the most frequently occurring labels.

3.2 Self-supervised Pre-training Model

The pre-trained model's architecture was built from scratch and was inspired by the BERT model described in section 2.6. For example, MLM was implemented to be able to benefit from self-supervised learning, and the encoder block in the backbone was designed in a very similar way as the one mentioned in section 2.5. The purpose of pre-training is to train a neural network on one task and dataset, e.g. predicting masked rows via MLM. The aim was to transfer the learned knowledge in the model onto a new task with possibly another, smaller, dataset.

Designing the pre-training model was a wide-ranging task. Partly due to the disparities between the data utilized by the original BERT model and the data employed in this thesis. Partly due to various possibilities for constructing the remaining components of the model, along with a multitude of parameters to fine-tune. So, in the following sections, the methodology to adapt MLM to the data in this thesis, and to optimize the design of all components of the pre-training model, will be explained in further detail.

3.2.1 Masked Language Modeling on Blink Feature Vectors

As the original architecture of Masked Language Modeling (MLM) is based on text data, it had to be customized to fit the data in this thesis, which is vectors containing continuous values. In the MLM there are three different actions that can be applied to the input data. One is to let it remain unchanged, one is to replace it with an arbitrary mask index, and the last one is to replace it with a random value. To let the data remain unchanged was thus the most trivial operation to translate since it could be done in the exact same way with different types of data. Furthermore, the operation where data is replaced with a mask index was rather straightforward as well. The operation simply replaces all the elements in the vector with the chosen mask index, which in this thesis was set to -1.

The operation when the data is replaced with a random value was however not as trivial to translate, due to the fundamentally different nature of the data in the two cases. When the operation is made on text data, the random value is a token chosen from a vocabulary available in the task, consisting of a finite number of tokens. Since the data in this thesis consists of continuous data, on the other hand, the random value can, theoretically, be set to an infinite choice of values. As the purpose of the action is to add noise as a regularization method, the decision was made not to replace values with totally random ones, but to tailor the operation to this specific task. Thus, when a vector of continuous values was chosen to be randomized, each element was multiplied with a noise factor, where the noise factor was chosen randomly but still within a chosen interval.

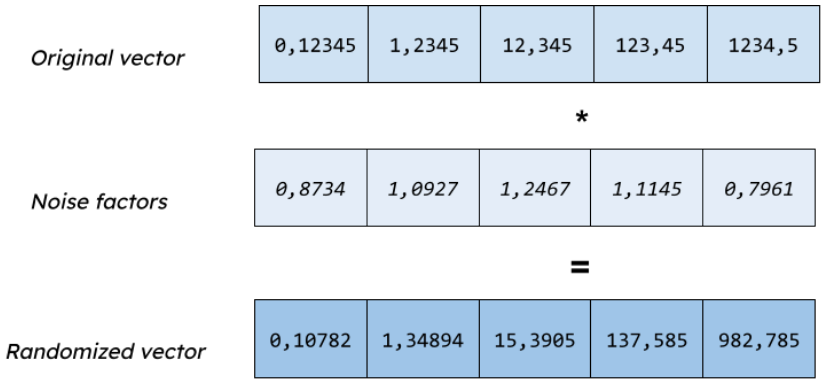

Figure 3.2: Randomizing vectors in MLM. Example of the operation where a vector of continuous values is randomized by being multiplicated with noise factors.

In summary, the MLM takes input data with shape (B, L, N), where B is the batch size, L is the sequence length, and N is the number of features. In each batch, 15% of the blinks in the sequence, i.e. rows, are chosen randomly to be masked with one of the three different masking options. The first option, to replace all features in the row with the masking index -1 is applied to 80% of the chosen rows. The second and third options, to randomize the features in a row with a noise constant and to leave a row unchanged, are applied to 10% each. The masked data is then returned, together with indices of which rows have been masked to be able to extract them for the prediction that will be done further on in the method. An example of how data can look before and after it's been processed in the MLM is shown in Figure 3.3.

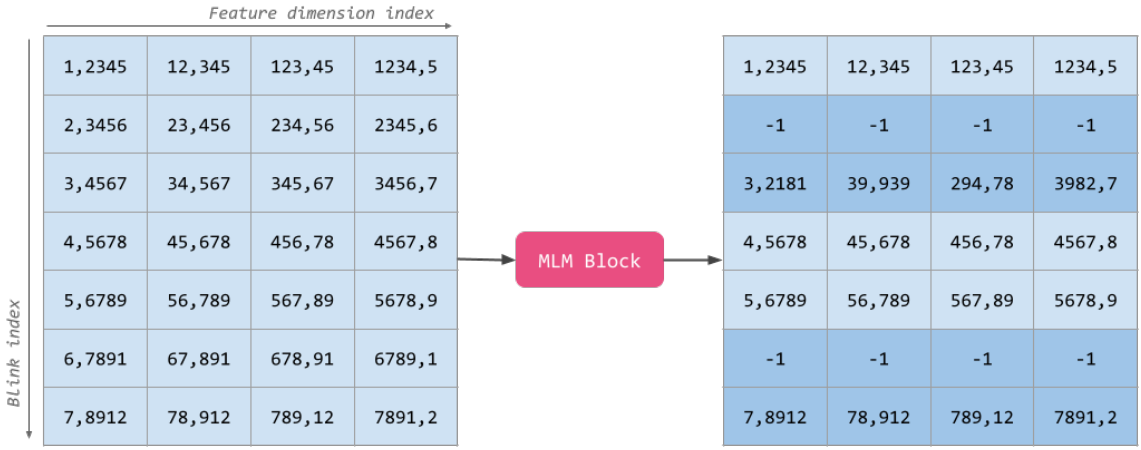

Figure 3.3: Data before and after MLM. Example of how data can look before and after being processed by the MLM block. The rows in darker blue are the ones that were handled and were either replaced with the mask index (-1), randomized with noise, or kept the same.

3.2.2 Upscaling Block

As mentioned in section 2.5.2, there are multiple different ways of performing feature upscaling depending on the task at hand. In this thesis, four different alternatives were evaluated, which were no upscaling at all, linear upscaling, and two different attention-based upscaling methods.

The first method, to not use any upscaling at all, is to simply add the positional embedding right onto the input data. This method naturally leads to a less complex network with fewer trainable parameters, and the embedding size being equal to the input size throughout the entire model. The second method, linear upscaling, is a bit more complex as it increases the number of trainable parameters and expands the embedded size.

The attention-based upscaling methods, however, are more complex in their nature. Both of them pass the input data through an encoder block with self-attention, to caption the relations within the data. The first method then upscales the encoded data with a linear layer, whilst the other one instead has a skip-connection of the input data which is upscaled with a linear layer, and concatenated with the encoded data.

All four of the methods were implemented and compared against each other to find the one that yields the best performance. How this was done will be explained in the next section.

3.2.3 Hyperparameter Sweep

As mentioned in section 2.8, it can be crucial to perform hyperparameter sweeps to find the optimal combination of settings of the hyperparameters. The process to find the optimized combination of hyperparameters in this thesis was made using the Python plugin Optuna, which is an automatic hyperparameter optimization software framework. By using Optuna, the sweep could be done in a more time-efficient and accurate way. It does however have limitations, e.g. GPU performance and memory, which restricted the scope of the search space.

The chosen search space is presented in Table 3.1, as well as the best value of each setting that was found. The sweep ran for 50 trials, where the first 25 of them were start-up runs, i.e. runs where the hyperparameters were chosen randomly from the search space and previous results weren't considered.

Table 3.1: Hyperparameter sweep of the pre-training model. The different hyperparameters in the pre-training model that were explored in the sweep, with their respective ranges or sets of values to explore, and finally the best combination of settings found.

Hyperparameter	Explored range or set	Best
Upscaling	[None, Linear, Attention+Linear, Attenion+Concatenation]	Attention+Linear
Embedding size	[32, 64, 128, 256, 512]	512
Encoder blocks	[2, 4, 8]	4
Attention-heads in encoder	[2, 4, 8]	4
Dropout *(Backbone)*	[0.0, 0.01, 0.02, ..., 0.2]	0.01
Hidden sizes *(Head)*	[[64, 64], [32, 32], [16, 16], [8, 8], [64], [32], [16], [8]]	[64]
Dropout *(Head)*	[0.0, 0.01, 0.02, ..., 0.2]	0.02
Batch normalization *(Head)*	[True, False]	True
Warmup epochs	[0, 1, 2, 3, 4, 5]	4
Learning rate	[0.00001, 0.00002, ..., 0.001]	0.00062
Accumulate gradient batches	[1, 2, 4, 8, 16, 32]	1
Decay factor	[0.85, 0.90, 0.95, 1.0]	0.9
Weight decay	[0.0, 0.0001, 0.0002, ..., 0.01]	0.0095
Gradient clip value	[1.0, 2.0, 4.0, 8.0]	1.0

Furthermore, the performed sweep also provided insight into how important the different parameters were for the performance, which is shown in Figure 3.4. The results show that the choice of upscaling method as well as the embedding size were the two parameters that had the biggest impact on the performance. Furthermore, the values of them that yielded the best result, i.e. attention-based upscaling and embedding size of 512, are, together with attention-based upscaling with concatenation, the most complex settings in the explored ranges.

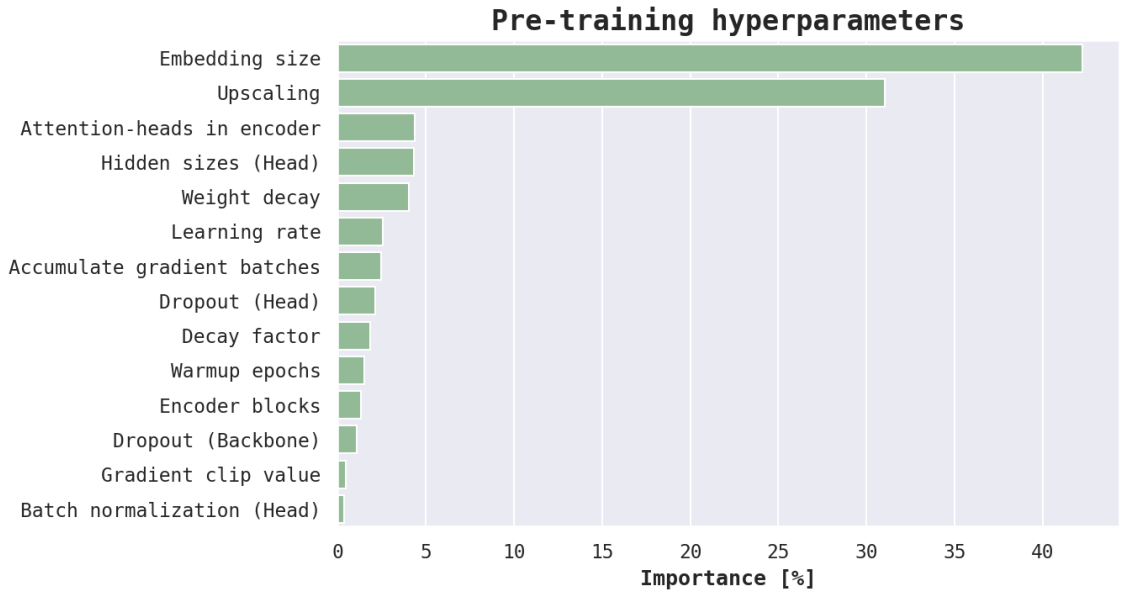

Figure 3.4: Importance of hyperparameters in the pre-training model. Bar chart of the importance of each hyperparameter included in the sweep session related to the pre-training model.

Finally, when the sweep was done, the final architecture of the pre-training model was formed, which will be described in greater detail in the following section.

3.2.4 Architecture of Pre-training Model

Based on the hyperparameter sweep, the final architecture of the pre-training model could be determined. As previously mentioned, the architecture was influenced both by the BERT model, and the encoder in [6]. From these architectures, some own implementations, like a customized MLM block and different upscaling methods, were made. To find the best combination of settings, a hyperparameter sweep was done. Based on the results from the sweep, the final pre-training model was designed.

In Figure 3.5, the architecture of the backbone is visualized. The chosen settings in the backbone's layers are the ones that were found in the sweep, presented in Table 3.1 in section 3.2.3. The initial input, which consists of batches with time series of blink feature vectors, is passed to the MLM block, escribed in section 3.2.1, to produce pseudo-labels. The output from the MLM block, which is of the same shape but with some rows masked, is then passed to the backbone. The data is processed through the backbone's upscaling block, and then, with positional embedding added to it, passed through four encoder blocks. Worth mentioning is that the attention layers are adjusted to not consider masked and padded rows. When the data has been processed through the backbone, it's returned as an output of the same shape as the initial input shape.

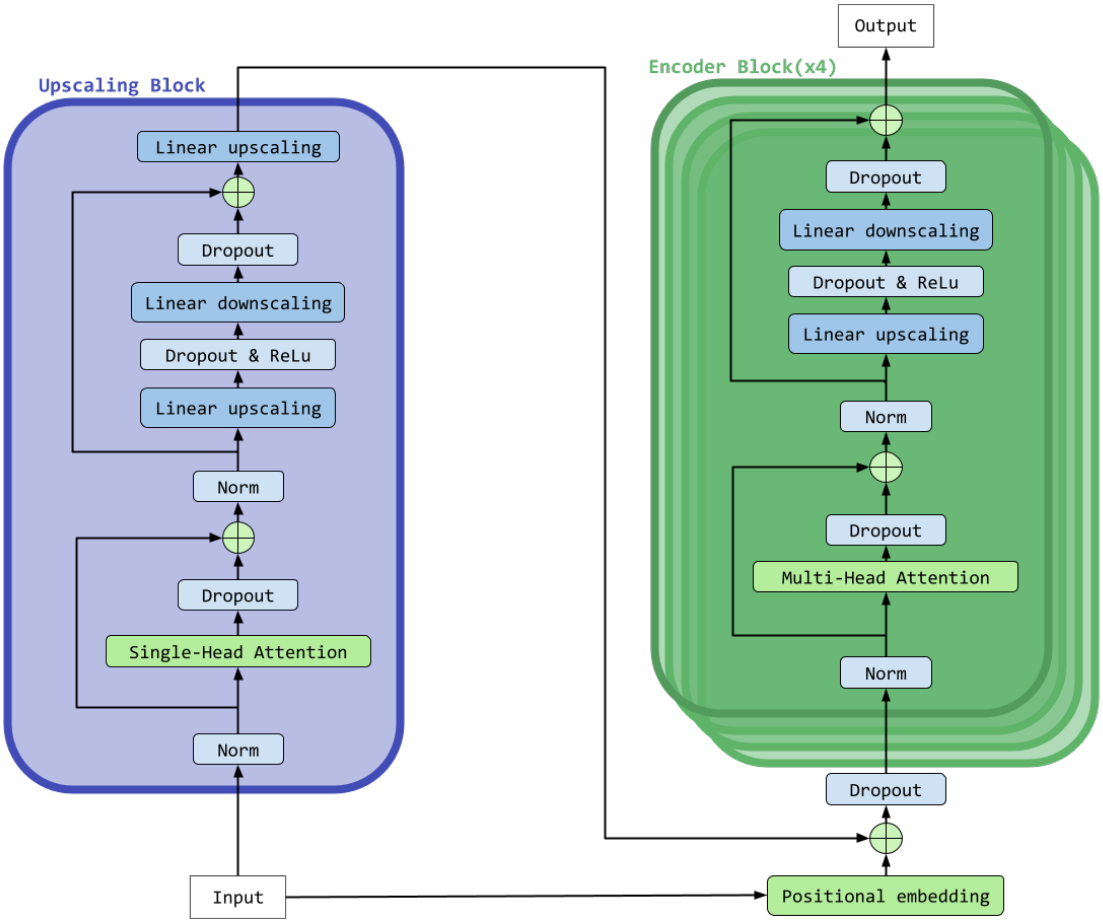

Figure 3.5: Backbone's architechture. Architecture of the backbone, with an upscaling block with self-attention and linear upscaling, positional embedding, and four identical encoder blocks.

After the data has been processed by the MLM block and the backbone of the pre-training model, it is passed on to the regression head. The regression head is responsible for performing the final prediction task, which in this case is to regress an output equal to the input data before it was masked. The head consists of several layers, which can be seen in Figure 3.6, and the hyperparameters of these layers are chosen based on the results from the hyperparameter sweep.

The output of the head will accordingly be of the same shape as the initial input that was fed to the MLM block. However, during the calculation of the loss function, only the rows that were masked in the MLM block are considered. By only considering the masked rows for the loss calculation, the model focuses on learning to predict the correct values for those specific rows.

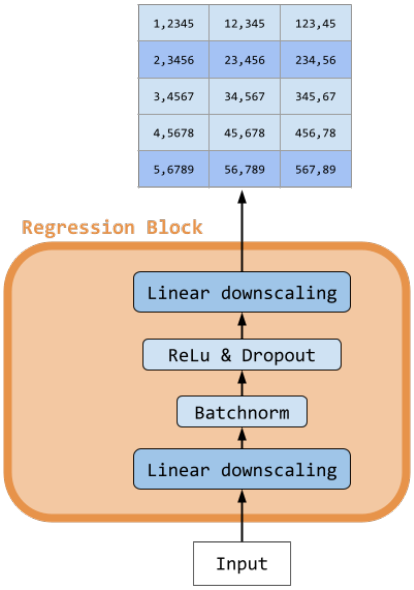

Figure 3.6: Pre-training model's head's architecture. Architecture of the head in the pre-training model, with an output with the same shape as the input in the head. The rows that were masked in MLM, in darker blue, are then filtered out from the output and used to calculate the loss.

3.2.5 Baseline Performance

Determining the performance of a model can be a non-trivial task, as a measured performance often is without context. So, in order to address the performance of the model in this thesis, some baseline performances were used for comparison. The baseline performances were simple solutions to the objectives, i.e. could be achieved without any advanced machine learning methods.

The objective of the pre-training model was to predict the masked rows of blink features produced during MLM. To assess the model's performance of predictions, two different baselines were chosen. The first one was to calculate the mean value of each feature in every sample, and the other one was to set the prediction to be equal to the next row in the sample. Both baselines were produced on the unmasked data, i.e. the calculated mean values were based on the entire sample, and the next row was always available. In a masked sample, the true mean can't be calculated due to the missing data, and since the masking is random, two adjacent rows could be masked which makes it impossible to use the next row as a prediction. Thus, the baselines did somehow have some advantages compared to the pre-training model since they didn't have to manage the masked rows but were generally simpler solutions.

3.3 Supervised Fine-tuning Model

The fine-tuning model was designed with the same general structure as the pre-trained model, i.e. a backbone in the form of an encoder (see Figure 3.5) followed by an output head. Similar to the pre-trained model there were a few required steps

to finish before being able to start training. The process included performing an additional hyperparameter sweep for the output head, exploring different options for transferring knowledge from pre-trained encoders, as well as evaluating the different encoder setups. The following sections will explain these steps in greater detail.

3.3.1 Hyperparameter Sweep

A similar hyperparameter sweep as the one in section 3.2.3 was set up for the fine-tuning model using the Optuna plugin. The difference though, was that it only considered parameters for the output head and some general setup variables since the specification of the encoder was given in the earlier hyperparameter sweep.

The search space for the selected parameters was the same as before, and the resulting parameters are presented in Table 3.2. Due to a smaller search space and a shorter compilation time per trial, the sweep ran for approximately 100 trials, but only the first 15 were start-up runs used for exploring randomly.

Table 3.2: Hyperparameter sweep of the fine-tuning model. The different hyperparameters in the fine-tuning model that were explored in the sweep, with their respective ranges or sets of values to explore, and finally the best combination of settings found.

Hyperparameter	Explored range or set	Best
Hidden sizes	[[64, 64], [32, 32], [16, 16], [8, 8], [64], [32], [16], [8]]	[16]
Dropout	[0.0, 0.01, 0.02, ..., 0.2]	0.16
Batch normalization	[True, False]	True
Accumulate gradient batches	[1, 2, 4, 8, 16, 32]	32
Gradient clip value	[1.0, 2.0, 4.0, 8.0]	1.0

As in the previous sweep, the result also contained information on the most important parameters for the evaluated model, which is presented in Figure 3.7. For the fine-tuning model, the most important hyperparameter was the choice of using batch normalization or not, followed by the hidden sizes. Moreover, as the figure shows, the best choices for these parameters were to use batch normalization and have a hidden size of 16.

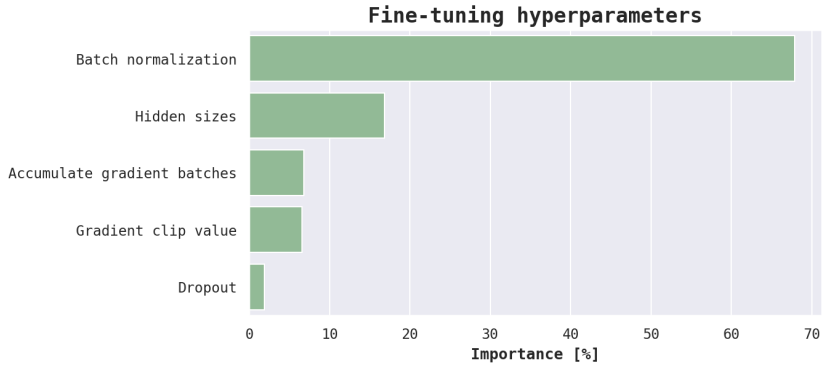

Figure 3.7: Importance of hyperparameters in the fine-tuning model. Bar chart of the importance of each hyperparameter included in the sweep session related to the fine-tuning model.

3.3.2 Architecture of Fine-tuning Model

As mentioned in section 3.3 the architecture of the fine-tuning model consisted of a backbone and a regression head. The regression head was designed based on the outcomes obtained from the hyperparameter sweep, while the backbone was re-implemented from the pre-training model, including the pre-trained weights.

The input data passed through the fine-tuning model was in the same shape as during pre-training. However, since the fine-tuning data included corresponding labels, it was not necessary to apply MLM to it to generate pseudo-labels. Instead, the input data was forwarded directly to the pre-trained backbone, and from its output, the last row, i.e. blink feature vector, of every sequence is selected as input to the fine-tuning model's regression head. Then, the selected blink feature vectors were forwarded through the head, which produced a continuous regressed value for each sample in the input, corresponding to a Karolinska Sleepiness Scale (KSS) value.

The selection of the last row from each sample sequence as the input data for the regression head, although somewhat arbitrary, could have been implemented in various ways. The key principle is however to ensure alignment with the target row(s) for accurate calculation of the loss.

3.4 Evaluation Methods

To answer the research questions, several runs were done with both the pre-training and fine-tuning model, to find the best performance together with the lowest cost, i.e. information quantity in the input data. The different runs mainly focused on varying the number of features and the amount of data utilized during the training processes. For instance, the pre-training model was trained with and without calibration features, to compare the impact of them. Furthermore, it was also trained with two different amounts of data, once with all data and once with a subset only

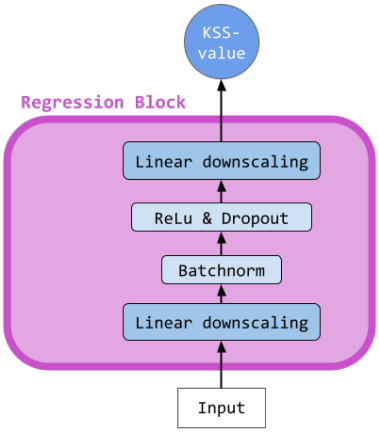

Figure 3.8: Fine-tuning model's head's architecture. Architecture of the head in the fine-tuning model, with a single KSS value for each sample in a batch as output.

containing half the amount of data. The subset was generated by selecting a percentage, in this case, 50%, of the drivers in the original dataset at random. The validation and test sets were however unchanged. By doing this, it was investigated how much data is needed to achieve the best possible results.

As mentioned in section 2.4 an important part of transferring knowledge from a pre-trained model is to initially freeze all layers, and then gradually unfreeze them. To find the best method for this, a set of runs were performed with different settings for unfreezing layers regarding both the number of layers and how long the layers were kept frozen. These runs were performed on the pre-training model trained on all data, and they explored the unfreezing of the four encoder blocks from the transferred pre-training model. Hence, the different unfreezing methods tested in this project ranged from unfreezing all four encoder blocks to keeping them all frozen during the entire fine-tuning process.

The process to choose a suitable unfreezing method also included experimenting with when to unfreeze layers, with setups from unfreezing after every third epoch to waiting for approximately 10 epochs. The reason for waiting for 10 epochs was because that was when the fine-tuning on the output head started to converge, meaning that the top layers of the model had adapted to the fine-tuning task. During these runs, the embeddings and positional encodings were kept frozen from the pre-training.

Two different pre-trained models were saved, i.e. the ones trained on different amounts of data, which then were used during the runs of the fine-tuning model. Initially, the fine-tuning model used the pre-trained model, trained on all data, to try different unfreezing methods as described above. The best-found unfreezing method was then used to train a fine-tuning model with learning transferred from the pre-trained model using only half of the data. By doing this, the amount of data during training could be compared to the results of the fine-tuning training as well. Furthermore, a fine-tuning model was also trained from scratch, i.e. without

any learning transferred from a pre-trained model, to investigate the pre-training impact on the performance.

4

Results

This chapter presents the results of the conducted experiments presented in section 3.4. Section 4.1 includes results on pre-training in relation to both the number of features and amount of data used during training, as well as comparisons to two different baseline performances. Furthermore, section 4.2 presents the fine-tuning results on different pre-trained encoders compared to a model trained from scratch, as well as different unfreezing methods.

4.1 Pre-training

The process of obtaining results began with the pre-training model, aiming to determine the optimal setup. This involved evaluating the inclusion of calibration features and the amount of training data for the model. Each subsequent experiment built upon the findings of the previous one.

The initial experiment focused on the impact of including calibration features in the pre-training data. As mentioned in section 3.1, Smart Eye devised a method to use these features to provide specific driver-related information. It was therefore important to investigate whether incorporating these calibration features would lead to meaningful improvements in model performance, and to what extent. Figure 4.1 shows that there was no significant change in performance when using the calibration features. Consequently, in the following experiments, the decision was made to exclude them, aiming to minimize the amount of input data used.

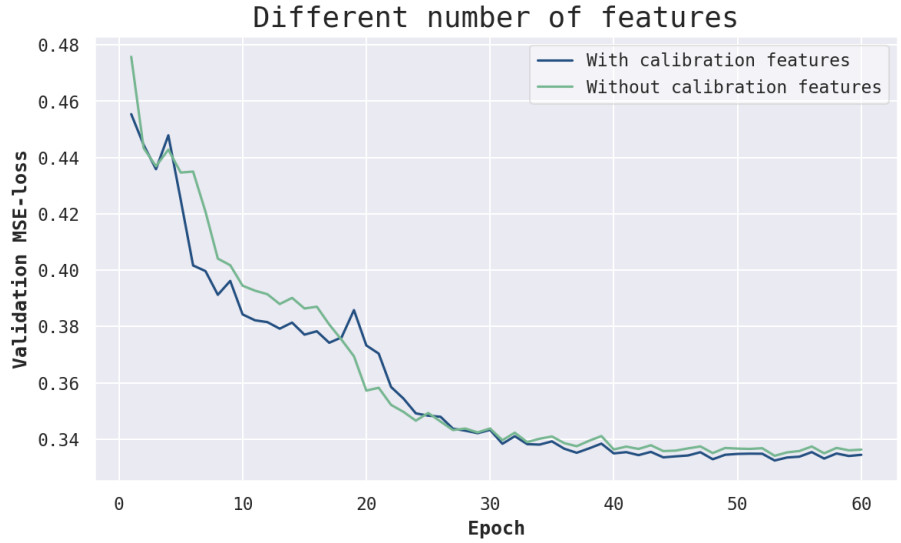

Figure 4.1: Pre-training on a different number of features. MSE loss during pre-training with and without calibration features.

The next experiment aimed to investigate the impact of the training data size on the performance of the pre-training phase. However, the ultimate objective was to evaluate how the amount of data used during pre-training would influence drowsiness prediction. Therefore, both models trained on different data sizes were utilized in the upcoming fine-tuning phase. The model was initially trained on the complete dataset, and its performance was compared to a model trained on a subset containing half the amount of data. The results, depicted in Figure 4.2, revealed a marginal difference in performance, with the pre-training conducted on the full dataset yielding slightly better results.

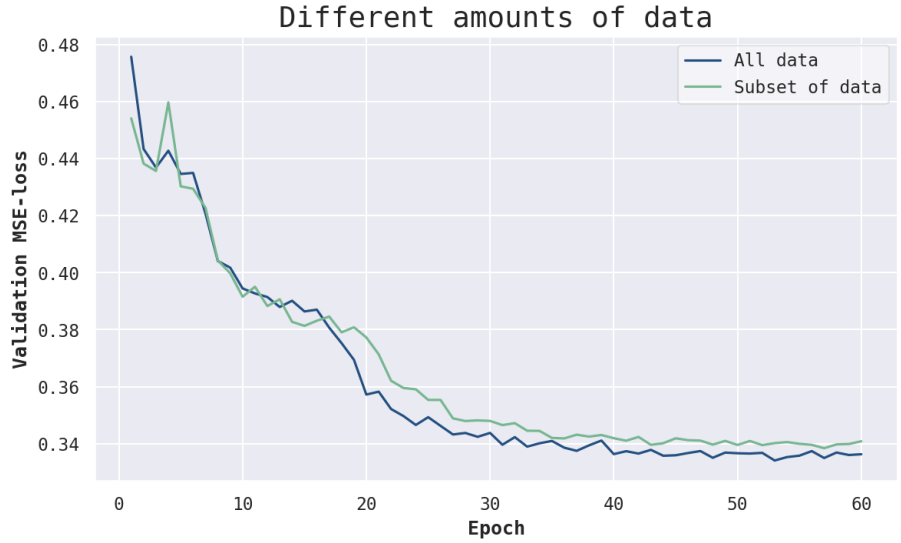

Figure 4.2: Pre-training on different amounts of data. MSE loss during pre-training without calibration features on all the data and on half of the data.

The last experiment carried out during the pre-training phase aimed to compare the performance of the best pre-training setup with two baseline approaches. The first baseline approach involved predicting the mean blink feature vector within each sample, excluding the masked rows. The second baseline approach involved predicting the next blink in the sequence (as explained in section 3.2.5). Figure 4.3 illustrates that the pre-trained model outperformed both baseline measures, demonstrating the best performance.

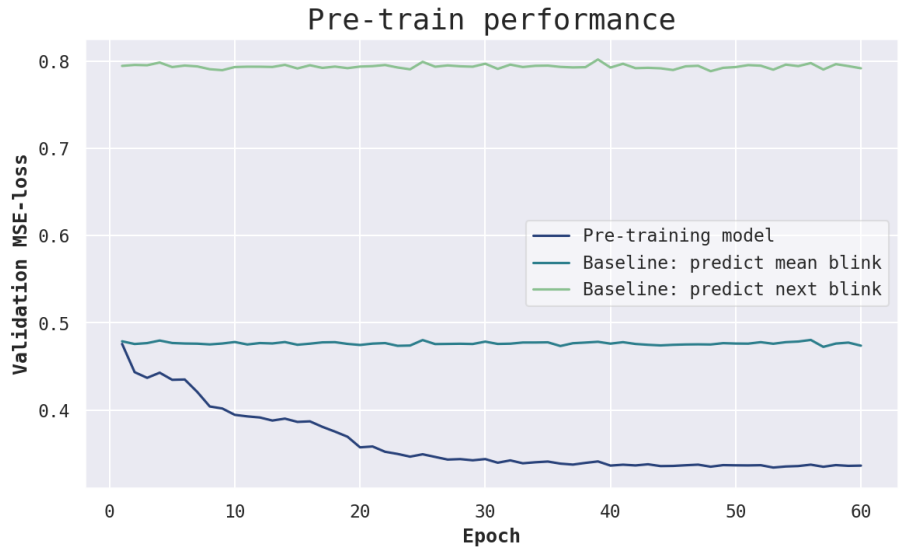

Figure 4.3: Pre-training performance compared to baseline. MSE loss during pre-training on all data compared to two baseline performances, i.e. the mean blink vector and predicting the next blink vector in the sequence.

4.2 Fine-tuning

As previously mentioned, the evaluation of the pre-training model provided two final pre-trained models, which were reused in the evaluation of the fine-tuning model. The first task was to test the different unfreezing methods, in order to find the one achieving best results.

As mentioned in section 3.4, the fine-tuning phase involved testing five different strategies for unfreezing the pre-trained model's backbone. In all cases, the fine-tuning model utilized the backbone from the pre-trained model, which was trained on the entire dataset. The performance of these strategies, measured by the 4-level KPI value during validation, is shown in Figure 4.4. The results indicated that the most successful method was to unfreeze the top two out of the four blocks. This unfreezing was performed after the third and sixth epoch, respectively. Consequently, the final model retained the frozen state of the initial layers throughout the training process, while the top two encoder blocks were fine-tuned to adapt to the new dataset and task.

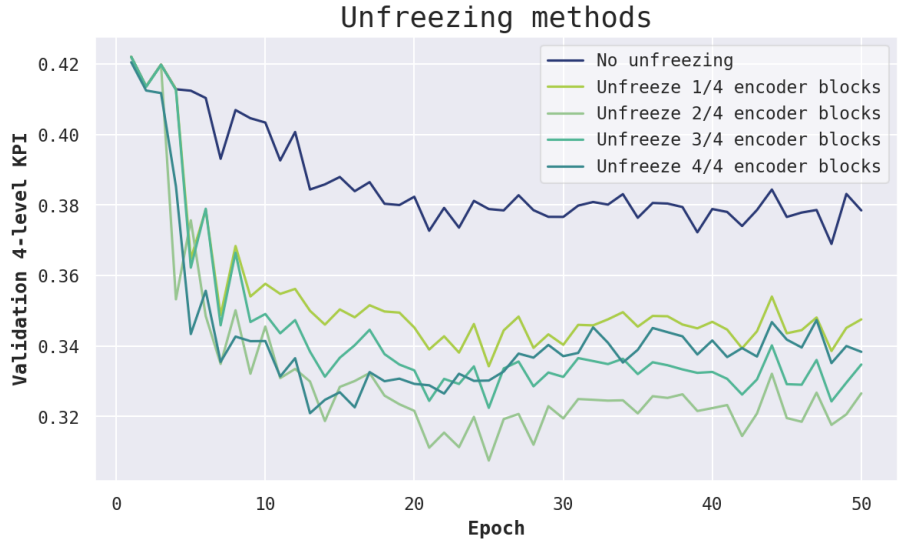

Figure 4.4: Unfreezing methods. Gradually unfreezing with different settings during fine-tuning on the model which was pre-trained on all data.

After selecting the unfreezing method, the fine-tuning process was conducted on the pre-trained models trained on different amounts of data, as well as on a model trained from scratch. Figure 4.5 shows that the fine-tuning of the pre-trained model trained on the entire dataset performed slightly better than the model trained on a subset of the data. However, the model fine-tuned from scratch exhibited the best performance, outperforming both pre-trained models. These results suggest that while pre-training can provide a beneficial starting point, fine-tuning a model from scratch may yield even better results.

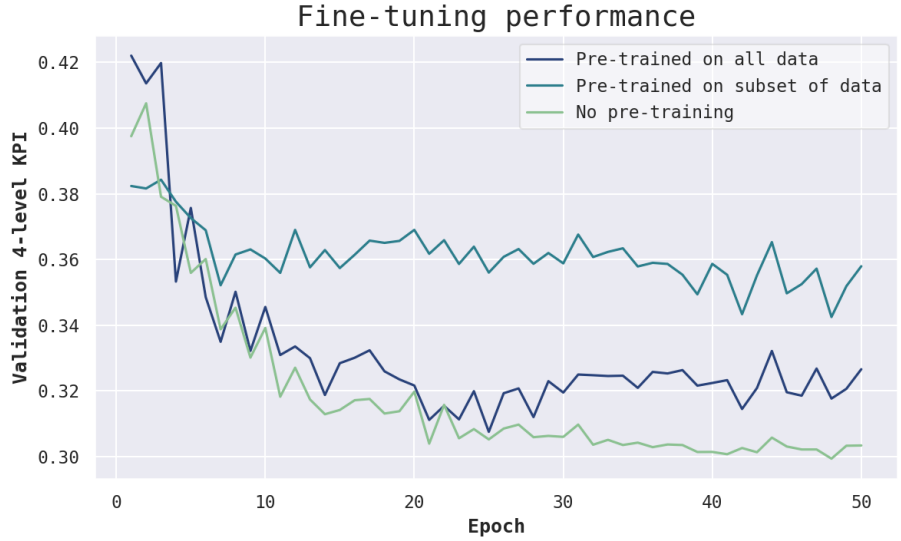

Figure 4.5: Performance of fine-tuning. The 4-level KPI validation performances for the three models, i.e. pre-training on all data, pre-training on a subset of data, and no pre-training at all.

The performance and accuracy of the models in estimating drowsiness levels, in the form of 4-level KSS values, can be observed through the confusion matrices shown in Figures 4.6 to 4.8. These matrices show the number of correctly and incorrectly predicted samples for each label. The KSS values, originally on a scale of nine, have been transformed into a 4-level scale for this analysis.

The confusion matrices on the left-hand side display the raw predictions, while the matrices on the right-hand side show the normalized results. Each cell represents the percentage of predictions for a specific label out of the total number of samples in that row.

Across all three models, the confusion matrices exhibit similar patterns, with label 2 being the most frequently predicted. The predictions do accordingly seem to be skewed to the left-hand side of the matrices.

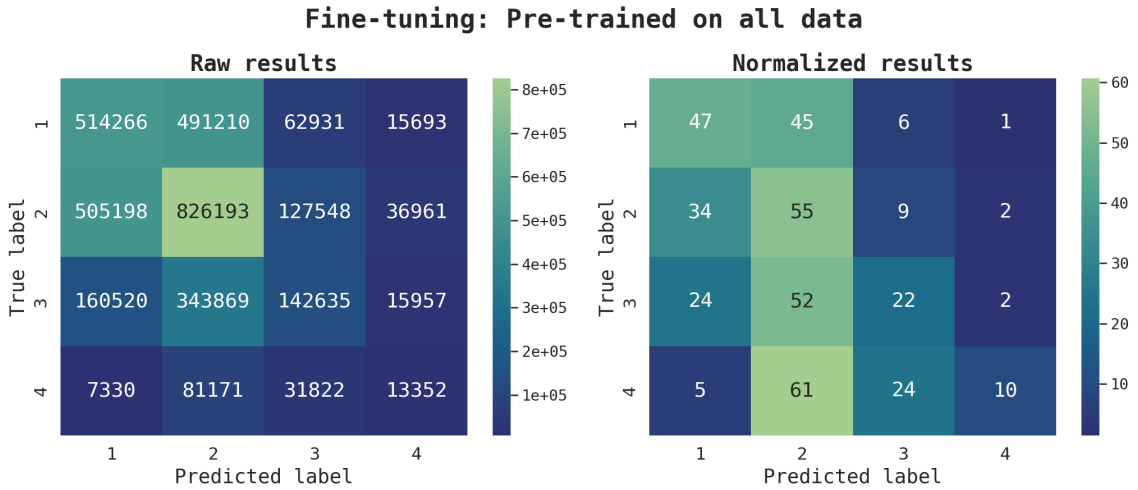

Figure 4.6: Confusion matrices when pre-trained on all data. Confusion matrices of the 4-level predictions when pre-training was done on all data. The one to the left is based on raw numbers of 4-level predictions, and the one to the right is normalized over the rows.

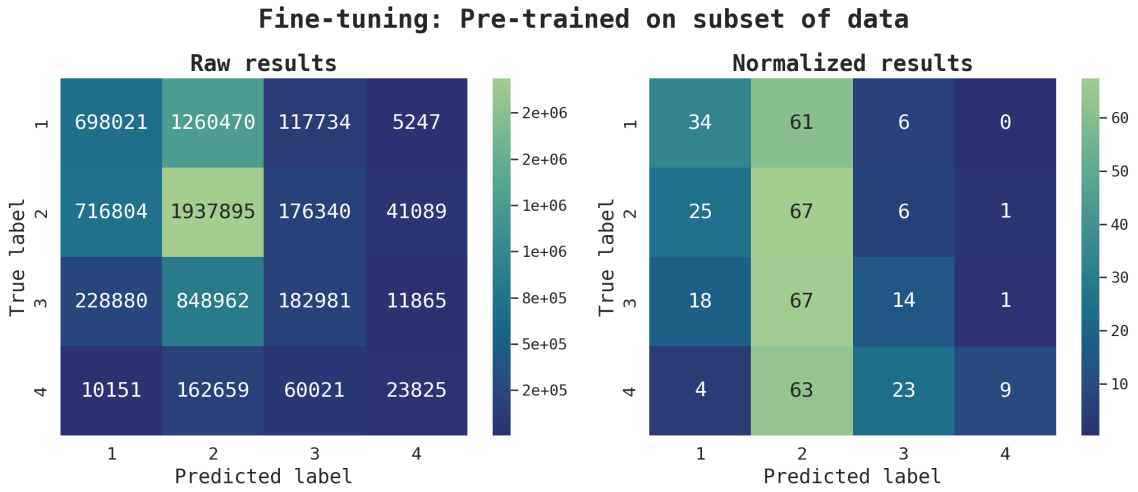

Figure 4.7: Confusion matrices when pre-trained on a subset of data. Confusion matrices of the 4-level predictions when pre-training was done on a subset of the data. The one to the left is based on raw numbers of 4-level predictions, and the one to the right is normalized over the rows.

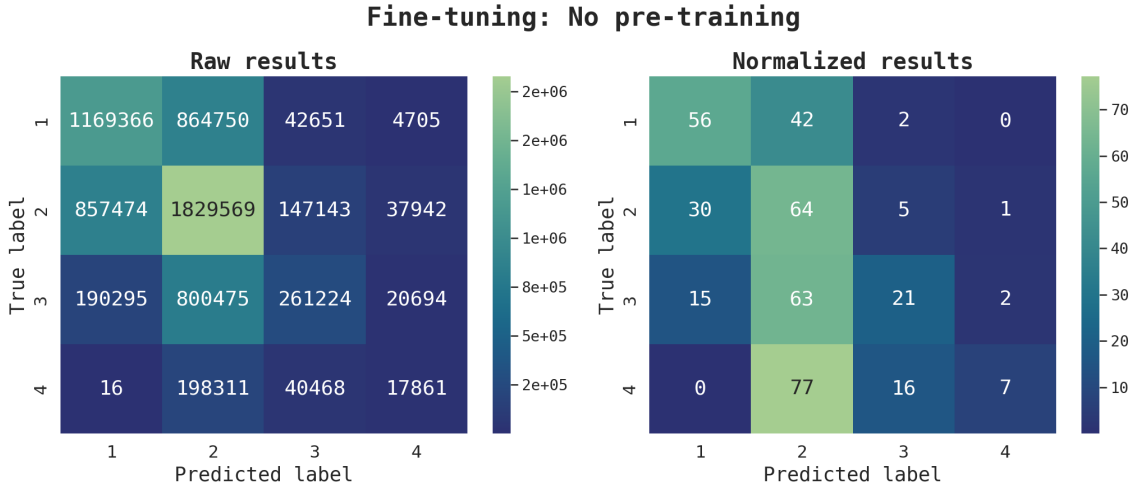

Figure 4.8: Confusion matrices when there was no pre-training. Confusion matrices of the 4-level predictions when no pre-training was done, i.e. the fine-tuning was done from scratch. The one to the left is based on raw numbers of 4-level predictions, and the one to the right is normalized over the rows.

5

Discussion

In this chapter, the results from Chapter 4 are discussed in relation to the research questions in section 1.3. The chapter is divided into three parts, one for each question, namely Self-supervised Pre-training, Drowsiness Prediction, and Amount of Data.

5.1 Self-supervised Pre-training

The primary aim of this thesis was to assess the influence of self-supervised pre-training on drowsiness prediction. To achieve this objective, a pre-training model was initially designed and evaluated. As illustrated in Figure 4.3 in section 4.1, the pre-training model outperformed both selected baseline performances. This outcome suggests that to a certain extent, predicting blink features in a sequence is feasible. It is however challenging to determine if the prediction is good enough, as the baselines are to some extent arbitrary. Nonetheless, the result shows that the model did not learn to predict the mean or next blink features, i.e. the baselines, in the sequence, but likely something more advantageous. Thus, it can be concluded that the model has acquired the ability to predict features to some extent.

There are several factors that might have affected the performance of the pre-training. One factor is the implementation of Masked Language Modeling (MLM). As mentioned in section 2.6, the original BERT model and techniques were developed with the aim of using text input for Natural Language Processing (NLP) tasks. Consequently, the MLM approach assumes the existence of a vocabulary including every possible word in the dataset. In this thesis, however, the data consists of continuous blink features, requiring adjustments to adapt the MLM method to the input data. This adaptation could be a contributing factor to the model's effectiveness in predicting blink features.

Another aspect that may have affected the result is the nature of the data itself. As previously mentioned, this thesis utilized continuous-valued blink features as opposed to the discrete vocabulary used by the BERT model. Predicting the precise target for multiple continuous values poses a great challenge, which might be more complex than the case of predicting discrete single targets.

Furthermore, the features in the data exhibited rather different behavior which could have contributed to the difficulties of predicting masked blinks. Although a detailed

description of the data is restricted by confidentiality concerns, certain aspects can still be addressed. For example, some of the features exhibit larger variance than others. Also, some features are more stable and show a more predictable behavior over time, while others are more volatile and show more fluctuating and unpredictable behavior over time.

During the construction of the pre-training model, a hyperparameter sweep provided optimal settings for the model in the span of approximately 50 runs. However, there are at least two essential aspects to take into consideration. Firstly, due to constraints invoked by GPU utilization and computation time, the number of runs had to be significantly decreased. Notably, Optuna, the plugin used for the sweep, typically recommends approximately 100-1000 runs to thoroughly explore the entire space of feasible combinations. Therefore, it might not be sufficient to only explore 50 runs to be able to find truly optimal parameter values for the pre-training model. Secondly, among the hyperparameters considered, the embedding size appeared as the most influential factor for pre-training performance, and the choice of size turned out to be the largest one. For that reason, it would have been interesting to experiment with even larger embedding sizes to assess the potential for further improvements in the pre-training performance.

5.2 Drowsiness Prediction

In section 4.2 the results from the fine-tuning are presented, where Figure 4.5 shows the 4-level KPI validation performances of two pre-trained models and one model without pre-training. Both pre-training models were optimized using various optimizing methods but differ in terms of the amount of data they were trained on. Despite the transfer of learning from pre-training in these two models, the model without pre-training achieved the highest performance. Thus, the results suggest that the self-supervised pre-training approach employed in this thesis does not appear to enhance drowsiness prediction.

There are several potential explanations for the lack of improvement resulting from pre-training. One explanation is the disparity between the pre-training task and the objective of fine-tuning. When examining the pre-training and fine-tuning tasks used in BERT, the similarities are more pronounced. For example, MLM and Next Sentence Prediction (NSP) during pre-training for NLP tasks provide token-level knowledge and information about the relationship between sentences. Consequently, fine-tuning tasks such as predicting the subsequent word or sentence stem from the pre-training tasks. In contrast, MLM during pre-training on blink feature data contributes blink feature-level knowledge within blink sequences. Although drowsiness can be inferred from blink features, the blink feature-level knowledge alone might be too disconnected from the relationship with drowsiness.

Another contributing factor to the result of drowsiness prediction could to some extent be the features and their connection to drowsiness. Due to delimitations, a detailed analysis of the features was not done. Hence, some of the features might be redundant. Some features could for example have a stronger connection to drowsi-

ness and thus provide vital information for accurate drowsiness prediction. By analyzing which features are the most relevant for the chosen pre-training and fine-tuning tasks it could possibly lead to an improved performance.

5.3 Amount of Data

The final research question investigates whether the quantity of data used during pre-training influences the performance of drowsiness prediction. Figure 4.2 illustrates the pre-training performance with different data amounts. Although there was no significant difference in performance observed, it is important to note that the focus was not specifically on assessing the impact during pre-training. However, as illustrated in Figure 4.5, it appears that utilizing a larger amount of data during pre-training can yield slightly better results.

Nevertheless, several contributing factors related to the available data for training should be taken into consideration. For instance, as mentioned earlier, the MLM technique randomly selects 15% of rows within a sequence. This introduces stochastic behavior in the predictions, as the same rows are unlikely to be chosen in subsequent epochs. Consequently, the data appears almost entirely new for each epoch, resembling an almost infinite amount of data, leastways pseudo-labeled data. This may result in the data being too diverse for the model to learn to represent all samples effectively.

6

Conclusion

The main objective of this thesis was to evaluate the efficiency of self-supervised pre-training in enhancing drowsiness prediction. A key subgoal was to assess the model's ability to predict blink features within a sequence. Additionally, the thesis aimed to investigate the impact of data volume used during pre-training, in regard to drowsiness prediction.

The pre-training process included an examination of feature selection and the utilization of calibration features. Calibration features were hypothesized to provide driver-specific information. However, the results revealed that the inclusion of calibration features did not significantly improve the performance of the pre-training task. Comparing the pre-training model's performance with two baselines, i.e. predicting the mean blink vector and predicting the next blink vector, indicated that it had learned certain representations more effectively.

In terms of the impact of data volume used during pre-training, no significant differences were observed at the pre-training stage. However, when fine-tuning the model for drowsiness prediction, using a larger volume of data during pre-training yielded better results. This suggests that exploiting Smart Eye's extensive collection of unlabeled data could be beneficial, as increased data volume positively influenced the model's performance.

The objective of improving drowsiness prediction through the implementation of self-supervised pre-training, however, was not achieved, as a non-pre-trained model exhibited better performance compared to its pre-trained counterpart.

The overall goal of this thesis was to evaluate whether Smart Eye could benefit from employing self-supervised pre-training on a large set of unlabeled data. Although the results did not provide conclusive evidence of pre-training's effectiveness in improving drowsiness prediction, it should not be entirely disregarded. The design and training of large neural networks involve various contributing factors, and this study represents an initial exploration of feasible methods to achieve Smart Eye's objectives.

6.1 Future work

The results obtained in this project did not provide definitive evidence either supporting or refuting the theory that self-supervised pre-training on blink features data could enhance drowsiness prediction with limited data. However, there are several areas that could be further explored to investigate this theory.

One interesting approach to explore is the implementation of Next Sentence Prediction (NSP), a technique demonstrated in the original BERT paper. By incorporating NSP as part of the pre-training process, it may be possible to gain valuable insights into the contextual relationships among blink feature vectors, potentially leading to improvements in drowsiness prediction. Additionally, it could be valuable to investigate if there are other pre-training tasks, beyond NSP, that are more directly aligned with the drowsiness prediction task.

Another aspect that requires further exploration is how the output data from the backbone should be propagated to the fine-tuning head. In this project, a single row from each sample in a batch was selected as input to the head. However, there are alternative approaches to consider, such as choosing multiple rows or utilizing all rows in the input representation. Experimenting with different strategies for passing information from the backbone to the fine-tuning head could provide valuable insights into the most effective approach.

Furthermore, it might be necessary to investigate the implementation of MLM in this project to ensure that it fits the data in an optimal way. One of the aspects to investigate is the masking process in the MLM. As of now, the method masks out complete rows. Another approach could be to handle the features separately, i.e. only masking out some of the features in a row. This could potentially enable the model to learn more about the different features individually, instead of complete rows. This could make the pre-training task easier for the model to learn since there is a possibility that some of the features are harder to adapt to and therefore are penalized more when updating the model throughout training.

References

[1] A. Bener, E. Yildirim, T. Özkan, and T. Lajunen, "Driver sleepiness, fatigue, careless behavior and risk of motor vehicle crash and injury: Population based case and control study," *Journal of Traffic and Transportation Engineering (English Edition)*, vol. 4, no. 5, pp. 496–502, 2017, ISSN: 2095-7564. DOI: https://doi.org/10.1016/j.jtte.2017.07.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S209575641730212X.

[2] E. Comission, *New rules to improve road safety and enable fully driverless vehicles in the eu*, 2022. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/ip_22_4312.

[3] C. Jacobé de Naurois, C. Bourdin, A. Stratulat, E. Diaz, and J.-L. Vercher, "Detection and prediction of driver drowsiness using artificial neural network models," *Accident Analysis & Prevention*, vol. 126, pp. 95–104, 2019, 10th International Conference on Managing Fatigue: Managing Fatigue to Improve Safety, Wellness, and Effectiveness"., ISSN: 0001-4575. DOI: https://doi.org/10.1016/j.aap.2017.11.038. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0001457517304347.

[4] E. Perkins, C. Sitaula, M. Burke, and F. Marzbanrad, "Challenges of driver drowsiness prediction: The remaining steps to implementation," *IEEE Transactions on Intelligent Vehicles*, pp. 1–20, 2022. DOI: 10.1109/tiv.2022.3224690. [Online]. Available: https://doi.org/10.1109%5C%2Ftiv.2022.3224690.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018. DOI: 10.48550/ARXIV.1810.04805. [Online]. Available: https://arxiv.org/abs/1810.04805.

[6] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2017. DOI: 10.48550/ARXIV.1706.03762. [Online]. Available: https://arxiv.org/abs/1706.03762.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV].

[8] I. Nasri, M. Karrouchi, K. Kassmi, and A. Messaoudi, *A review of driver drowsiness detection systems: Techniques, advantages and limitations*, 2022. DOI: 10.48550/ARXIV.2206.07489. [Online]. Available: https://arxiv.org/abs/2206.07489.

[9] S. M. Belz, "An on-road investigation of self-rating of alertness and temporal separation as indicators of driver fatigue in commercial motor vehicle operators.," 2000.

[10] A. Christer, F. Carina, A. Anna, and H. David, *Video-based observer rated sleepiness versus self-reported subjective sleepiness in real road driving*, 2015. DOI: 10.1007/s12544-015-0188-y. [Online]. Available: https://doi.org/10.1007/s12544-015-0188-y.

[11] A. Shahid, K. Wilkinson, S. Marcu, and C. M. Shapiro, "Stanford sleepiness scale (sss)," in *STOP, THAT and One Hundred Other Sleep Scales*, A. Shahid, K. Wilkinson, S. Marcu, and C. M. Shapiro, Eds. New York, NY: Springer New York, 2012, pp. 369–370.

[12] T. Åkerstedt and M. Gillberg, "Subjective and objective sleepiness in the active individual," *International Journal of Neuroscience*, vol. 52, no. 1-2, pp. 29–37, 1990, PMID: 2265922. DOI: 10.3109/00207459008994241. eprint: https://doi.org/10.3109/00207459008994241. [Online]. Available: https://doi.org/10.3109/00207459008994241.

[13] B. Mehlig, *Machine Learning with Neural Networks*. Cambridge University Press, Oct. 2021. DOI: 10.1017/9781108860604. [Online]. Available: https://doi.org/10.1017%5C%2F9781108860604.

[14] A. K. Kuchibhotla, L. D. Brown, A. Buja, and J. Cai, *All of linear regression*, 2019. DOI: 10.48550/ARXIV.1910.06386. [Online]. Available: https://arxiv.org/abs/1910.06386.

[15] L. Ericsson, H. Gouk, C. C. Loy, and T. M. Hospedales, "Self-supervised representation learning: Introduction, advances, and challenges," *IEEE Signal Processing Magazine*, vol. 39, no. 3, pp. 42–62, May 2022. DOI: 10.1109/msp.2021.3134634. [Online]. Available: https://doi.org/10.1109%5C%2Fmsp.2021.3134634.

[16] J. Yao, Q. Wu, Q. Feng, and S. Chen, *Learning downstream task by selectively capturing complementary knowledge from multiple self-supervisedly learning pretexts*, 2022. arXiv: 2204.05248 [cs.LG].

[17] Y. Wang, D. Sun, K. Chen, F. Lai, and M. Chowdhury, *Egeria: Efficient dnn training with knowledge-guided layer freezing*, 2023. arXiv: 2201.06227 [cs.LG].

[18] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, *How transferable are features in deep neural networks?* 2014. arXiv: 1411.1792 [cs.LG].

[19] M. E. Peters, M. Neumann, M. Iyyer, *et al.*, *Deep contextualized word representations*, 2018. DOI: 10.48550/ARXIV.1802.05365. [Online]. Available: https://arxiv.org/abs/1802.05365.

[20] C. Song and A. Raghunathan, *Information leakage in embedding models*, 2020. arXiv: 2004.00053 [cs.LG].

[21] S. Kundu, H. Mostafa, S. N. Sridhar, and S. Sundaresan, *Attention-based image upsampling*, 2020. arXiv: 2012.09904 [cs.CV].

[22] Y. Liu, M. Ott, N. Goyal, *et al.*, *Roberta: A robustly optimized bert pretraining approach*, 2019. arXiv: 1907.11692 [cs.CL].

[23] A. Jadon, A. Patil, and S. Jadon, *A comprehensive survey of regression based loss functions for time series forecasting*, 2022. DOI: 10.48550/ARXIV.2211.02989. [Online]. Available: https://arxiv.org/abs/2211.02989.

[24] Z. Vujovic, "Classification model evaluation metrics," *International Journal of Advanced Computer Science and Applications*, vol. Volume 12, pp. 599–606, Jul. 2021. DOI: 10.14569/IJACSA.2021.0120670.

[25] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24, Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[26] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, 2019. arXiv: 1907.10902 [cs.LG].

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

CHALMERS
UNIVERSITY OF TECHNOLOGY