

Time-Series Classification with Uni-Dimensional Convolutional Neural Networks

*An Experimental Comparison with Long
Short-Term Memory Networks*

Sharanan Kulam



Thesis submitted for the degree of
Master in Programming and Networks
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2019

Time-Series Classification with Uni-Dimensional Convolutional Neural Networks

*An Experimental Comparison with Long
Short-Term Memory Networks*

Sharanan Kulam

© 2019 Sharanan Kulam

Time-Series Classification with Uni-Dimensional Convolutional Neural Networks

<http://www.duo.uio.no/>

Printed: Reprosentralen, University of Oslo

Abstract

In recent years, research in machine intelligence has gained increased momentum, where neural network models have made significant contributions in various fields, like image classification and language understanding. Recurrent neural networks (RNNs) are often the preferred approach for tasks like language understanding and time-series analysis. However, a known problem is their inefficiency to capture long-term dependencies, giving rise to alternative RNNs. Long short-term memory (LSTM) and gated recurrent units (GRU) solve this problem, but on the expense of computational effort. As a result, convolutional neural networks (CNNs) have been explored for sequence modelling in recent years and shown to outperform RNNs in general. This efficiency, however, is examined only by a few comparative studies, where most primarily focus on language tasks. Similar studies are far more absent in the time-series classification domain, where traditional methods are often used.

To address this shortcoming and further understand the effects of CNNs and RNNs in the time-series classification domain, we evaluate two shallow networks in this thesis, a CNN and an LSTM. We extend the few existing comparisons through an experimental approach and provide a baseline comparison of both for the time-series classification domain, where such studies are almost absent.

To do so, we created an easily extensible system for running experiments and evaluated our models on three different datasets using cross-validation. We classify depressed patients using motor activity, predict the energy demand of Electric Vehicles (EVs) and classify readiness of football players. The system was used to evaluate CNN and LSTM separately for each dataset and is generalisable for multiple neural network models that can be used for similar comparative studies.

We show that simple CNN achieves the same performance as LSTM and is faster to train. For two of our use cases, CNN is more than 30 times faster in terms of seconds used, but we see a trade-off between training time used in seconds and iterations, as CNN uses more training iterations. We conclude that for time-series classification, CNNs should be the preferred choice over LSTM, because of their effectiveness in performance and faster training.

Acknowledgements

A hectic year has passed, and at last, a long journey comes to an end. This thesis became everything else than initially intended, but it widened my perspectives. My passion for research has become stronger. First and foremost, I want to send my gratitude to my supervisors, Michael Riegler and Pål Halvorsen. Thank you for your time, availability and all help throughout the thesis, it is much appreciated. I deeply admire your work, and being part of Simula Research Laboratory has been truly inspiring and motivational. Thank you for the opportunity. I wish you the best and hope your research reach out to as many as possible. Your societal impact, especially in the health domain, is of great value. Also, you nurture a great research environment at the lab, especially for master students. I am grateful to have been part of that.

Secondly, thank you to all MSc and PhD students at the lab. The lunches, conversations, late evenings and discussions were enjoyable. Mathias, Edvarda, Thanh Son, Asad, Hanna, Steven and Vajira, thank you! Although our paths diverged suddenly, I genuinely appreciate the days together at the lab! All the best in your professional careers as well!

To my dearest friends; it has been great knowing you these years and thank you for being there. My inner circle is small, and your importance to me as friends can not only be described in words. Deep Singh, Anh Tin Nguyen, Thanh Son Vo; grazie mille! Here is to many more years ahead.

To my dear parents; your work ethic, unconditional love and values are respected qualities. Your work ethic and grit have taught me many things. Experiences and views you share have also widened my perspectives of life in general. To that, and all the privileges you have given me, I'm infinitely thankful. This is for you.

Infinite gratitude is challenging to squash into one page. To all others who proofread the thesis, supported me, helped me in any way or whom I forgot to mention here, thank you. Your contributions do not go unnoticed.

Plato is my friend - Aristotle is my friend - but my best friend is truth.

Isaac Newton
in *Certain Philosophical Questions*

Contents

List of Figures	vi
List of Tables	viii
List of Equations	ix
1 Introduction	1
1.1 Background and motivation	1
1.2 Problem statement	4
1.3 Limitations and scope	4
1.4 Research method	5
1.5 Contributions	6
1.6 Outline	7
2 Background	9
2.1 Related work	9
2.1.1 An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling	9
2.1.2 Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline	10
2.1.3 Deep learning for time series classification: a review	10
2.1.4 Forecasting Economics and Financial Time Series: ARIMA vs. LSTM	11
2.1.5 Summary of related work	11
2.2 Machine Learning	12
2.2.1 Supervised Learning	12
2.2.2 Unsupervised Learning	13
2.2.3 Short on the curse of overfitting	13
2.2.4 Summary of machine learning	13
2.3 Neural Networks	13
2.3.1 Definition	14
2.3.2 The learning process	15
2.3.3 Summary of artificial neural networks	19
2.4 Recurrent neural networks	19
2.4.1 Definition	20
2.4.2 The learning process	21
2.4.3 The difficulty of learning	21
2.4.4 Summary of recurrent neural networks	22

2.5	Convolutional neural networks	22
2.5.1	Definition	22
2.5.2	The learning process	23
2.5.3	Summary of convolutional neural networks	24
2.6	Summary of background	25
3	Methodology	27
3.1	Overview	27
3.1.1	Data preparation and data analysis	29
3.1.2	Initial experiments and optimised experiments	29
3.2	System specifications	29
3.2.1	Tensorflow	30
3.2.2	Keras	31
3.2.3	Pandas	31
3.2.4	Experiment framework	32
3.3	U_1 : Depression detection	34
3.3.1	Medical background	36
3.3.2	Dataset	37
3.3.3	Summary of use case	40
3.4	U_2 : Energy prediction	41
3.4.1	Energy background	41
3.4.2	Dataset	42
3.4.3	Summary of use case	48
3.5	U_3 : Football readiness classification	48
3.5.1	Football performance background	48
3.5.2	Dataset	49
3.5.3	Summary of use case	54
3.6	Model development	55
3.7	Model training and hyperparameters	56
3.7.1	Model training	56
3.7.2	Hyperparameter selection	58
3.8	Experiment evaluation	61
3.8.1	Evaluation methods	61
3.8.2	Evaluation metrics	63
3.9	Summary of methodology	65
4	Experiments	67
4.1	U_1 : Depression detection	67
4.1.1	Experiment overview	67
4.1.2	Results for LSTM	69
4.1.3	Results for CNN	79
4.1.4	A comparison on LSTM and CNN	88
4.1.5	Summary of experiments	94
4.2	U_2 : Energy prediction	94
4.2.1	Experiment overview	94
4.2.2	Results for LSTM	96
4.2.3	Results for CNN	100
4.2.4	A comparison on LSTM and CNN	103

4.2.5	Summary of experiments	104
4.3	U_3 : Football readiness classification	105
4.3.1	Experiment overview	105
4.3.2	Results for LSTM	108
4.3.3	Results for CNN	113
4.3.4	A comparison on LSTM and CNN	118
4.3.5	Summary of experiments	121
4.4	Discussion	121
4.4.1	Faster training times	122
4.4.2	The importance of data	123
4.4.3	The effect of model architecture	123
4.4.4	Application areas and use cases	125
4.5	Summary and overview of all experiments	126
5	Conclusion	129
5.1	Summary	129
5.2	Contributions	130
5.3	Future work	131
	Bibliography	141
	Appendices	145
A	Background	145
A.1	Overfitting in machine learning	145
B	Experiments	147
B.1	Depression: second initial CNN result	147
B.1.1	Moving average of training history	147
B.1.2	Confusion matrix for best performing model	147
B.2	LSTM: additional experiments in energy prediction use case	147
B.3	CNN: additional experiments in energy prediction use case	147
B.4	CNN: additional experiments in football use case	148
B.5	Table summary of all experiments	148
C	Methodology	155
C.1	Example of model building in Keras	155
C.2	Example of Pandas data manipulation operations	156
C.3	Configuration file used in thesis	156
C.4	Execution script used in thesis	156

List of Figures

1.1	16-hour average energy prices and consumption	2
1.2	Patiently activity levels from the depression dataset [91]	3
1.3	Experiment overview and design	5
2.1	A simple neural network with 2 hidden layers	14
2.2	Commonly known activation functions	17
2.3	Illustration of the recurrent unit	20
2.4	Illustration of convolution in CNNs	24
2.5	Illustration of pooling in CNNs	25
3.1	Method overview	27
3.2	Method detail	28
3.3	Tree structure of experiment framework	33
3.4	Illustration of experiment framework	34
3.5	Tree structure of generated logs and results	35
3.6	U_1 , general overview of activity levels	38
3.7	U_1 , activity heatmap from the depression dataset	39
3.8	U_1 , temporal correlations for activity levels	40
3.9	U_2 , battery level heatmap from the EV dataset	44
3.10	U_2 , temporal correlations for battery levels	45
3.11	U_2 , histogram of battery levels	46
3.12	U_2 , median level driven distances	47
3.13	U_3 , average correlation matrix	51
3.14	U_3 , cross-correlation function, readiness	52
3.15	U_3 , distribution of readiness scores	53
3.16	General overview of model architecture	55
3.17	K-fold partitioning	62
3.18	Confusion matrices	62
4.1	U_1 - depression use case, experiment overview	68
4.2	U_1 , LSTM validation history for I_1	70
4.3	U_1 , LSTM validation history for I_2	71
4.4	U_1 , LSTM validation history for I_3	73
4.5	U_1 , LSTM validation history for R_1	74
4.6	U_1 , LSTM validation history for R_2	76
4.7	U_1 , LSTM validation history for R_3	77
4.8	U_1 , CNN validation history for I_1	80
4.9	U_1 , CNN validation history for I_2	81

4.10	U_1 , CNN validation history for R_1	83
4.11	U_1 , CNN validation history for R_2	84
4.12	U_1 , CNN validation history for R_3	85
4.13	U_1 , CNN validation history for R_4	86
4.14	U_1 , total training time for models	89
4.15	U_1 , summary of evaluations	90
4.16	U_1 , histogram for activity levels and KDE	92
4.17	U_2 - energy use case, experiment overview	95
4.18	U_2 , LSTM validation history for I_1	97
4.19	U_2 , LSTM validation history for I_2	98
4.20	U_2 , LSTM validation history for R_1	99
4.21	U_2 , CNN validation history for I_1	101
4.22	U_2 , CNN validation history for R_1 and R_2	102
4.23	U_2 , total training time for models	104
4.24	U_2 , summary of evaluations	105
4.25	U_3 - football use case, experiment overview	107
4.26	U_3 , LSTM validation history for R_1	109
4.27	U_3 , LSTM best performing model for R_1	110
4.28	U_3 , LSTM validation history for R_2	111
4.29	U_3 , LSTM best performing model for R_2	112
4.30	U_3 , CNN validation history for I_1	114
4.31	U_3 , CNN validation history for R_1	115
4.32	U_3 , CNN validation history for R_2	116
4.33	U_3 , total training time for models	118
4.34	U_3 , summary of evaluations	119
4.35	U_3 , confusion matrices	120
4.36	Average training times for all experiments	122
4.37	Average training iterations for all experiments	124
4.38	Evaluation summar of all experiments	126
A.1	Illustration of overfitting	145
B.1	Depression CNN: initial result 2, moving average	148
B.2	Depression CNN: initial result 2, confusion matrix	149
B.3	LSTM: Additional optimised results, EV use case	150
B.4	CNN: Additional initial results, EV use case	151
B.5	CNN: Additional initial results, football use case	152

List of Tables

3.1	Overview of system specifications	30
3.2	Overview of depression dataset	37
3.3	Overview of EV dataset	42
3.4	Overview of PMSys dataset	50
3.5	General overview of trainable model parameters	56
3.6	Overview of model hyperparameters	59
3.7	Overview of model hyperparameters	59
3.8	Overview of model hyperparameters	60
4.1	U_1 , LSTM initial hyperparameter configurations	69
4.2	U_1 , LSTM cross-validation average for I_1	70
4.3	U_1 , LSTM cross-validation average for I_2	71
4.4	U_1 , LSTM cross-validation average for I_3	72
4.5	U_1 , LSTM initial hyperparameter configurations for optimised results .	74
4.6	U_1 , LSTM cross-validation average for R_1	75
4.7	U_1 , LSTM cross-validation average for R_2	76
4.8	U_1 , LSTM cross-validation average for R_3	78
4.9	U_1 , LSTM weighted average summary	78
4.10	U_1 , CNN initial hyperparameter configurations	79
4.11	U_1 , CNN cross-validation average for I_1	80
4.12	U_1 , CNN cross-validation average for I_2	81
4.13	U_1 , CNN initial hyperparameter configurations for optimised results .	82
4.14	U_1 , CNN cross-validation average for R_1	82
4.15	U_1 , CNN cross-validation average for R_2	84
4.16	U_1 , CNN cross-validation average for R_3	86
4.17	U_1 , CNN cross-validation average for R_4	87
4.18	U_1 , CNN weighted average summary	87
4.19	U_2 , LSTM initial hyperparameter configurations	96
4.20	U_2 , LSTM cross-validation average for I_1	96
4.21	U_2 , LSTM cross-validation average for I_2	97
4.22	U_2 , LSTM hyperparameter configurations for optimised result	99
4.23	U_2 , LSTM cross-validation average for R_1	99
4.24	U_2 , CNN initial hyperparameter configurations	100
4.25	U_2 , CNN cross-validation average for I_1	101
4.26	U_2 , CNN weighted average summary	103
4.27	U_3 , LSTM hyperparameter configurations for R_1	108
4.28	U_3 , LSTM cross-validation average for R_1	109
4.29	U_3 , LSTM hyperparameter configurations for R_2	111

4.30	U_3 , LSTM cross-validation average for R_2	112
4.31	U_3 , LSTM weighted average summary	113
4.32	U_3 , CNN initial hyperparameter configurations for I_1	114
4.33	U_3 , CNN cross-validation average for R_1	116
4.34	U_3 , CNN cross-validation average for R_2	117
4.35	U_3 , CNN weighted average summary	117
B.1	Table summary of all experiments	153

List of Equations

2.1	The simple linear regression model	14
2.2	The multiple linear regression model	15
2.3	The activation function for the j -th node	16
2.4	The threshold activation of the j -th node	16
2.5	The residual sum of squares	16
2.6	The cross-entropy loss	18
2.7	The batch gradient descent	19
2.8	The recurrent unit	20
3.1	The accuracy metric	63
3.2	The precision metric	63
3.3	The specificity metric	64
3.4	The sensitivity metric	64
3.5	The F1 metric	65
3.6	The MCC metric	65
4.1	Multilabel classification formulation for football usecase	106

Chapter 1

Introduction

RNNs have gained increased popularity in recent years due to their efficiency in sequential modelling problems. While various architectures like LSTM and GRU have achieved state-of-the-art results, mostly in language processing tasks, the rise of CNNs for sequential problems have emerged in recent years. This thesis research the use of CNNs with a comparison against LSTM in the time-series domain, a field where traditional methods have usually been used. Motivated by the recent trends of CNNs on language problems, we aim to understand the potential application area on time-series classification. Opposed to other comparative studies with language tasks as default benchmarks, we analyse the application on three use cases in the medical domain, energy domain and sports domain.

1.1 Background and motivation

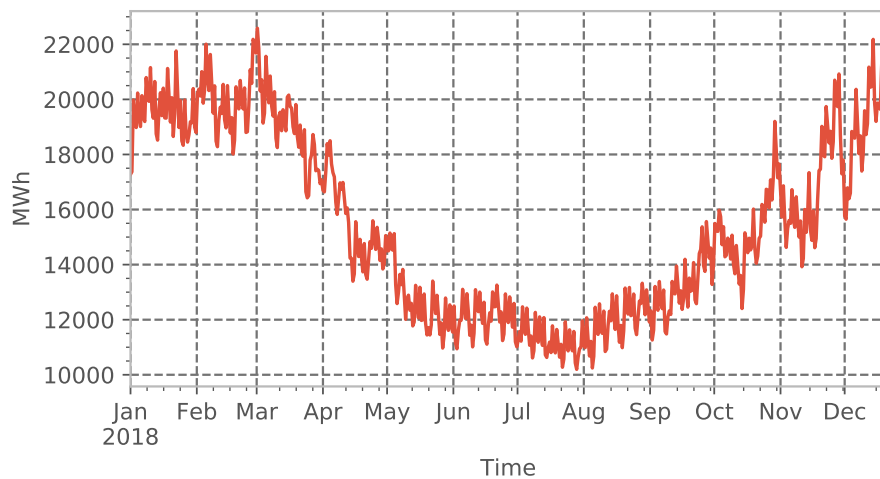
A time-series is an ordered collection of data points where each data point is an observation in time [2, 82]. A series with observations on multiple variables at any given timestep t is known as a multivariate series. On the other hand, a univariate time-series includes observations on one variable over time. For instance, temperature measurement is considered univariate. If other variables like humidity, wind and cloud coverage are included, the series is multivariate. **Figure 1.1** shows two different time-series of energy prices and energy consumption in Norway, whereas **Figure 1.2** shows motor activity levels of patients from one of our use cases.

Time-series analysis is a field concerned with the analysis and development of models that capture the underlying temporal dependencies in time-series data. Modelling temporal data is important in many fields like econometrics, financial analysis, weather forecasting and medicine [82] and enables a possibility of understanding the future based on historical observations [51, 83]. Some examples include:

- **Energy consumption** - Understanding future energy consumption is relevant for power generators in order to prevent excessive power production that increases production costs.
- **Weather forecasting** - Multivariate time-series analysis can be used to forecast



(a) 16-hour average of energy prices in Norway.



(b) 16-hour average of energy consumption in Norway

Figure 1.1: The figures illustrate two different time-series of 16-hour average energy prices and consumption in Norway. The data used to generate this figure is accessible through Nordpool [25]

the weather, which depends on multiple factors, including the temporal element, like time of the day or month of the year.

- **Anomaly detection** - Detecting inconsistencies and anomalies in time-series can be important to prevent disastrous outcomes. Detecting failures in industrial components or finding anomalies in network traffic are some examples.

Although there are various application areas, it is common to apply time-series analysis to develop predictive models that take time dependencies into account. Conventional modelling approaches often assume time-independent observations [82] and do not work equally well for time-series data. However, various approaches have been trending in the field of Machine Learning (ML) in recent years, with the development of sequential models like Recurrent Neural Networks (RNNs). They are one type of neural network models that take temporal dependencies into account

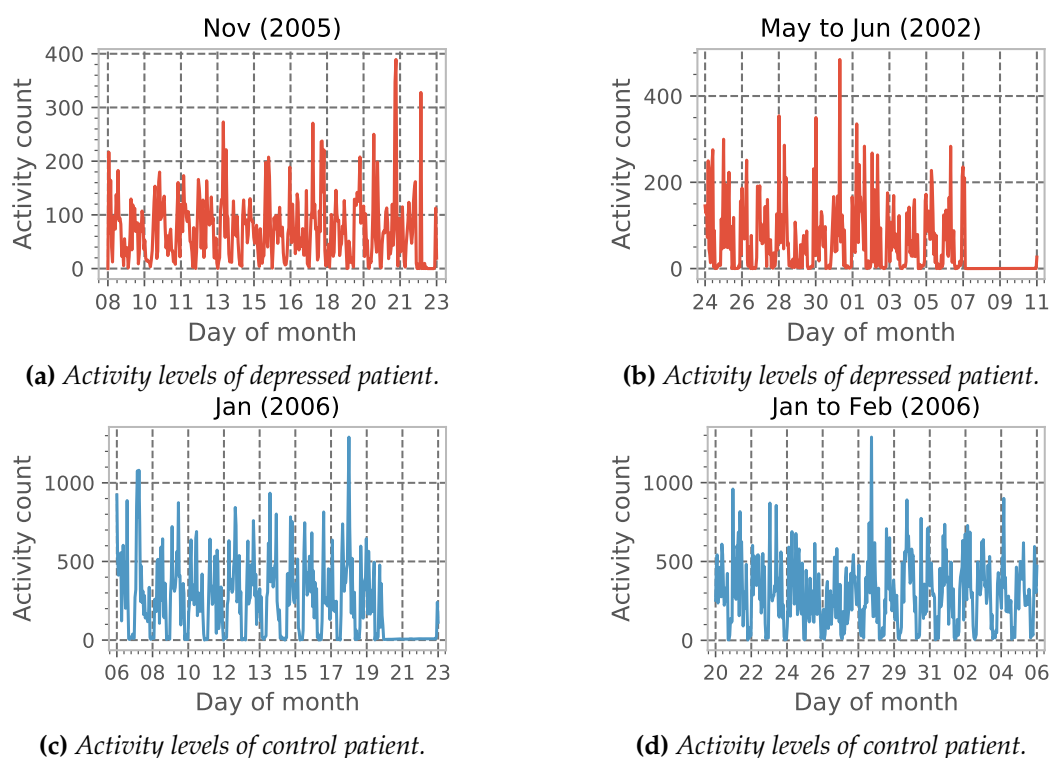


Figure 1.2: Patient activity levels from the depression dataset [91]. The plots show hourly average activity levels for 2 depressed (top row) and 2 control patients (bottom row). Note that all series are not from the same period.

and have had successful applications in later years. Many applications is in the field of language processing [14, 34, 35, 79, 97], but another example is self-driving cars [6].

While the examples mentioned above are naturally sequential problems, they are not time-series. As described earlier, traditional methods are often applied to modelling time-series. Most approaches use simple regression analysis or more extensive methods, like autoregressive (AR), autoregressive moving average (ARMA) or state-space models [82]. With gained popularity around neural network models like RNNs, the potential application area of such models to time-series is only recent [26, 44, 84] or relatively limited. Fawaz et al. [26] present a review on the application of deeper neural networks to time-series classification, which is one of few extensive studies that look at the application of neural network models to time-series.

Despite the rise of RNNs and limited application to time-series, there are still certain shortcomings of this kind of models. Various RNN architectures have different shortcomings. The vanilla RNN has an architecture that makes it challenging to learn longer temporal dependencies. In order to overcome this problem, so-called gated architectures were introduced, like Long Short-Term Memory-based networks (LSTMs) and Gated Recurrent Units (GRUs). However, these architectures are more complex and require extensive computational effort [8, 16, 67].

Although LSTM and GRU have proven to be efficient, research on different neural network architectures to sequential problems has been explored, like Convolutional Neural Networks (CNNs). CNNs are one type of network architecture mostly known

for applications in the image analysis and image processing domain [4, 39, 52, 70] because of their ability to preserve spatial information. However, they have proven to be applicable to sequential tasks like machine translation and sentence classification [24, 48, 49] as well.

Furthermore, the studies of RNNs are often benchmarked across various language processing tasks. Moreover, the empirical evaluations of CNNs, which is relatively limited on sequential problems, also focus on language tasks [5, 48, 49]. In the time-series domain, such comparative studies are not extensive, and the application of RNNs and CNNs in the field and predictive modelling has only emerged recently [12, 56, 78, 93]. Because of this lack in comparative studies and benchmarks in the domain of time-series classification, it is difficult to determine how CNNs are preferable. The thesis further aims to explore this through an experimental study to provide empirical evidence on the application of CNNs on time-series classification with a comparison against LSTM.

1.2 Problem statement

As we have described, there are limited comparative studies of RNNs and CNNs in the time-series domain, where traditional statistical models are often used. While RNNs and CNNs have been evaluated on time-series data, research in the same domain is fairly limited. To better understand the application of alternative architectures like CNN for the time-series domain, we aim to answer the following research question:

RQ: How do CNNs compare to LSTMs for time-series classification?

To answer this question, we compare CNN and LSTM on three different use cases across the medical-, energy- and sports domain. Moreover, our proposed research question further motivates for understanding the following two sub-questions:

1. *SQ1: How does LSTM perform for time-series classification?*
2. *SQ2: How does CNN perform for time-series classification?*

Our sub-questions are implied in *RQ*. The aim of *SQ1* and *SQ2* is to answer how the architectures perform in the context of the classification tasks. The overall aim of this thesis is to show how CNNs compare to LSTMs for time-series classification in general. Hopefully, the work can provide a good baseline for future comparative studies and also empirical evidence on the application of CNNs to time-series classification.

1.3 Limitations and scope

Based on the outlined problem statement, the scope of this thesis is to research the effect of uni-dimensional CNNs for time-series classification and provide both a systematic and experimental comparative study against LSTM. For reliable and empirical evaluations, we test both architectures on three different use cases.

Furthermore, within this scope, we develop a system that runs experiments automatically. We limit this system to be specific enough in the context of our proposed research question, that is, comparing LSTM and CNN, but general enough to be extended for later use.

We decided to limit this comparative study to LSTM and CNN. First of all, because gated architectures have shown to be more promising in various sequential modelling tasks in recent years. Second, because of time and resource constraints as the topic of this thesis changed over time, and extensive comparisons became less practical. Although GRU is another gated architecture, we chose LSTM due to its popularity.

Furthermore, in a comparative study like this, understanding the underlying mechanisms and performance of each architecture requires lowering the complexity of the models. We decided to keep our models shallow to easily compare the performance of both in the classification tasks and against each other.

1.4 Research method

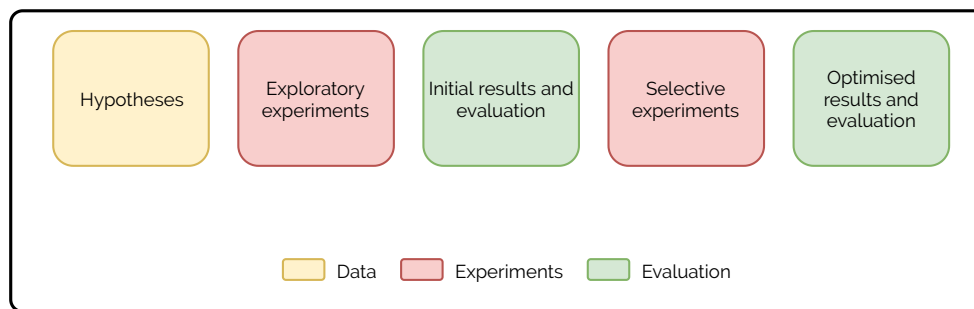


Figure 1.3: General level experiment design. Hypotheses formed during the data analysis are explored iteratively through exploratory experiments with multiple configurations. The initial results and evaluations are used to perform specific experiments in order to try to achieve optimised results, by configuring more specific hyperparameters.

Within the field of Computer Science, which is a relatively new field of study, there are various existing research methods. Some approaches may include both theoretical and experimental methods [37]. In this thesis, we apply the research method in the report *Computing as a discipline* [21], proposed by the Association for Computing Machinery (ACM) in 1989. The report divides the discipline of computing into three distinct paradigms; (i) theory, (ii) abstraction/modelling and (iii) design. **Figure 1.3** shows a general-level design for the experiments in this thesis, and shows how the two former paradigms fit into this scheme.

Theory - The theory paradigm is rooted in the discipline of mathematics. The paradigm is concerned with the development of a coherent and valid theory. ACM divides this paradigm into four steps; (i) characterise objects of study (definition), (ii) hypothesise possible relationships among them (theorem), (iii) determine whether the relationships are true (proof) and (iv) interpretation of results.

In this thesis, the objects of study is a comparison of LSTM and CNN for time-series

classification. While doing so, we discuss different neural network architectures like recurrent- and convolutional architectures, their shortcomings and benefits, and how they differ in terms of application domains. We perform multiple experiments to understand how one architecture performs compared to the other and determine their effectiveness based on the results.

Abstraction - The abstraction/modelling paradigm is rooted in the experimental scientific method. The report describes this is a four-step process when researching and investigating a phenomenon; (i) form a hypothesis, (ii) construct a model and make a prediction, (iii) design an experiment and collect data and (iv) analyse results.

The thesis provides an extensive comparison with multiple experiments. First, the hypotheses are formed based on data analysis for each use case. The design of experiments is based on the data analysis, and the initial hypotheses form based on this. Upon experiment execution, we try different hyperparameters, in which the results are analysed separately for each use case. Both models are compared on a use case level, before a general-level discussion and analysis of results is presented.

Design - The design paradigm is the last paradigm rooted in engineering principles, which is described as a four-step process. The process is concerned with the construction of a system or device that solves a given problem; (i) state requirements, (ii) state specifications, (iii) design and implement the system and (iv) test the system.

This thesis has a theoretical approach in the comparison of two algorithms. To compare both, we develop an experimental framework that facilitates for comparison of LSTM and CNN on our use cases. An additional requirement is that the system enables the opportunity to extend to other models or use cases in the future. The implemented framework is general enough to be used in research projects where comparisons of neural network algorithms are relevant. With minor modifications, the framework can be used with other use cases and datasets, in addition to other model architectures.

1.5 Contributions

The main contribution of this thesis is a systematic comparison of uni-dimensional CNNs and LSTM on various time-series classification tasks, a field where such studies are relatively limited, and explored by only a few. In general, our work can be attributed to our primary research question and summarised as follows:

Our empirical evidence suggests that a simple CNN can achieve at least the same results as an LSTM, and in many cases, it is marginally better. CNN is faster to train in terms of seconds used, where we see a speedup factor of more than 30 for two out of three use cases. However, we also see that LSTM is more efficient in learning underlying data distributions and uses less training iterations to do so. This trade-off, however, is marginal in which we conclude that CNNs should be the preferred choice over LSTM, because of their effectiveness in performance and faster training.

Moreover, the proposed models are tested on three use cases. First, we classify depressed patients based on motor activity levels [29, 91] in a binary classification problem. With existing baselines on 73% and 0.43 for accuracy and MCC, respectively,

our CNN achieves an accuracy of almost $80\% \pm \sim 2\%$ across all results, and an MCC of 0.54 on average. Comparably, our LSTM achieves an average accuracy of $82\% \pm \sim 1.5\%$ and an MCC of 0.59 at its best. Additionally, CNN outperforms LSTM in terms of the number of seconds used for training, being 46 times faster.

Second, we predict the energy demand for Electric Vehicles (EVs) in a multiclass classification problem using a smaller dataset to showcase the potential in this domain. Applications of neural networks in this domain are limited. However, we have acquired a smaller dataset and provide a simple baseline for future research. We report an average accuracy of 60% and an MCC of 0.04 for CNN. Comparably, LSTM achieves an accuracy of 58% and MCC of -0.01. Moreover, both models use approximately the same training time in seconds. Overall this showcase the potential improvement area in the field of EV charging optimisation.

Lastly, we classify the readiness of football players [69, 94] in a multilabel classification problem. We predict four qualitative features describing the readiness of football players, namely, mood, stress, soreness and fatigue. Overall, we report average accuracies of almost 89% for our CNN and 90% for LSTM, in which the CNN is 35 times faster when comparing the number of seconds used for training.

To perform this comparative study, we also developed an experimental framework, which automatically configures and builds, compiles, trains our models and runs experiments and stores results accordingly. We used the Python programming language and Keras framework for model development. Overall, it was developed specifically to the use cases and models in this thesis but is flexible enough to be used for similar comparative studies of neural network models, with minor modifications. Our system can be separated into three parts, where we first set initial hyperparameter configurations manually and specify use cases and model architecture. We then run the experiment pipeline based on these configurations which automatically builds and trains the models, before running the evaluation pipeline at last, which stores various experiment results, training history, evaluations, confusion matrices, metrics and other statistics required for evaluation. These evaluations are saved to persistent storage automatically, based on the given configurations.

1.6 Outline

The current chapter, **Chapter 1**, introduces a general level background and motivation, presents the research question, research method and lastly, the scope and various limitations. Overall, the outline of the thesis separates into five chapters.

Chapter 2 - Background: We present the necessary background in the theoretical part of the thesis. We discuss machine learning, artificial neural networks, recurrent neural networks and convolutional neural networks. They are defined in separate sections, where each presents a definition, the learning process in each architecture and related work within the context of the research question of this thesis.

Chapter 3 - Methodology: We present our methodological approach. The first part of this chapter gives a general level overview of the methodology and how it underpins

the problem statement. Further on, we present system specifications and overview of our experiment framework. We then discuss each use case, present the dataset and perform data analysis, before explaining the model training process and chosen hyperparameters. At last, we discuss our evaluation methods and metrics before summarising the methodology in the last section.

Chapter 4 - Experiments: We present the results for each use case. For each use case, we give an overview of the experiment design before presenting results for LSTM and CNN. Further on, we present a discussion and comparison in the context of each use case. We then summarise our findings for each use case. At the end of this chapter, we provide a general-level discussion and a summary of all experiments and results.

Chapter 5 - Conclusion: We summarise this thesis, our findings and conclude our contributions. At last, we discuss the potential future work of our presented work.

Chapter 2

Background

This chapter presents the related work and theoretical background. In the first section, we present related work of comparative studies of RNNs and CNNs. We then define the concepts in machine learning and give an overview of supervised learning and unsupervised learning. Further on, we present the general concept of artificial neural networks and describe the various underlying mechanisms of training these, followed by definitions and overview on recurrent neural networks and convolutional neural networks, which build on the same concepts.

2.1 Related work

While the popularity of RNNs and CNNs have increased in recent years, many of their common applications are found in language processing [14, 35, 79] and image classification tasks [38, 43, 52, 88], respectively. Although traditional models like autoregressive/moving average models like ARIMA [2, 82] and Hidden Markov Models (HMM) [47, 78] are often used for time-series, there has been some research in recent years looking at the effect of RNNs [26, 28, 44, 83, 84] and CNNs [12, 56, 78, 93]. While some studies explore CNNs and RNNs on time-series separately, hybrid approaches have been explored as well [17, 93]. However, there are limited studies on the comparison of both architectures, and less so in the time-series domain. Some of the following research papers aim to provide relevant baselines for such comparisons.

2.1.1 An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

Of all the literature we have looked at, this study presents the most extensive comparison of RNNs and CNNs. In their paper, Bai, Kolter, and Koltun [5] present an empirical comparison of different RNN architectures against CNN on various sequence modelling tasks. They address the question of whether the recent successes of CNNs on sequential problems [48, 66] are specific to the studied application domains, or if they are applicable to sequence tasks in general. In their paper, they

study the effect of CNNs on various problems in which they conclude how CNNs are indeed applicable to different domains.

Their CNN, named TCN for Temporal Convolutional Network, outperforms canonical recurrent architectures like LSTM and GRU. Through the experiments, they use the same configurations with various kernel sizes and layers. Dilated convolutions are used with Adam-optimiser and a learning rate of 0.002. For the RNNs, automatic hyperparameter optimisation is applied, where they use grid search to find optimal configurations. Moreover, the study further analyses the effect on how the TCN captures temporal patterns and how the memory mechanism works. Overall, they find that the CNN-based network shows the ability to capture long history more efficiently than the RNNs as well.

While this comparison study is essential to our research, the effect on time-series is not explored in their paper. This further motivates for our proposed research question, in which we extend the study to understand how CNNs can be used for time-series classification.

2.1.2 Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline

This paper explores the application of CNNs for time-series classification. In their study, Wang, Yan, and Oates [93] present a comparison of their Fully Convolutional Neural Network (FCN) to various traditional time-series classification models. Their proposed FCN is, in essence, a CNN without pooling in the hidden layers and applies global average pooling instead of using a dense optimisation layer in the last layer. Additionally, two neural network models are compared as well, the first being a vanilla neural network (MLP), whereas the second is a residual neural network (ResNet)[38]. The models are tested across 44 distinct time-series datasets, in which their study aims to present a baseline for neural network-based models in the time-series classification domain.

Their results suggest CNNs are indeed applicable to time-series in which the compared models are outperformed across almost all tested use cases. Further on, they also show how their FCN, which is shallow with only three layers, outperforms architectures like ResNet. Overall, Wang, Yan, and Oates [93] show how shallow CNNs can effectively be applied to time-series classification, although a comparison against recurrent architectures is not present.

2.1.3 Deep learning for time series classification: a review

Fawaz et al. [26] presents an extensive empirical comparison in their paper, in which they aim to explore different deep neural network architectures and find the current state-of-the-art for time-series classification. Overall, they train 8730 deep learning models on 97 time-series datasets, the most extensive to date. In their paper, their comparison includes the vanilla architecture (MLP) and different convolutional architectures like FCN/CNN, (used by Wang, Yan, and Oates [93]) and ResNet. However, they include one type of RNN in their comparative study, namely Time

Warping Invariant Echo State Networks (TWIESN), although we consider this out of the scope of this thesis.

While all of the tested networks are relatively shallow, with anything between three and eleven layers, the overall findings suggest that deep learning models can achieve state-of-the-art performance with architectures like FCN and ResNet in the time-series classification domain. In the context of our proposed research question, the compared recurrent model, TWIESN, achieved competitive results to the convolutional architectures in many cases. However, the convolutional models seemed to work most efficient overall, which further poses an interesting question of whether this applies to LSTMs as well.

2.1.4 Forecasting Economics and Financial Time Series: ARIMA vs. LSTM

In this paper, Siami-Namini and Namin [83] study the effect of LSTM on forecasting of financial time-series where they compare the model against the more traditional ARIMA. The aim of their research is to present a comparison of ARIMA and LSTM, in which they explore the questions (i) *which algorithm, ARIMA or LSTM, performs more accurate of time series data?* and (ii) *Does the number of training times in deep learning-based algorithms influence the accuracy of the trained model?.* They evaluate both models across twelve different datasets with a train/test split of 70/30, ranging from 200 to 600 monthly observations and one dataset with 1200 weekly observations. Their results suggest that LSTM is efficient across all use cases with an overall average reduction of approximately 85% in RMSE, which is the metric they use to evaluate the models. Although their study shows that increased iterations do not contribute to better performance, they showcase how traditional approaches like ARIMA is outperformed.

2.1.5 Summary of related work

In this section, we presented relevant literature. The first study by Bai, Kolter, and Koltun [5] is one of few, which provides an empirical evaluation on the comparison of CNNs and RNNs, but focus on different language tasks. Further on, the second paper explored the effects of CNNs on time-series classification, in which the models were compared to traditional time-series classification models. The third paper provides a similar study and is more extensive, in which they explore various CNNs to find the current state-of-the-art. However, one shortcoming of this study is their lack of RNNs as baseline comparisons. The last study compares LSTM and provides a more systematic review with a comparison against ARIMA, showing how neural networks can outperform traditional time-series models in general.

Nevertheless, from the presented literature, it is evident that there is an absence of comparative studies of RNNs and CNNs, in which the few existing ones focus on various language tasks. Moreover, such studies are almost non-existent in the field of time-series classification, where current studies, like those presented, mostly focus on comparisons of RNNs or CNNs against traditional time-series classification models.

2.2 Machine Learning

Machine Learning (ML) is a field in computer science concerned with developing data-driven computer algorithms/systems. This class of algorithms can learn from observations and data, without explicit instructions or where explicit instructions are not conventional. It is a cross-disciplinary field that combines techniques from optimisation, statistics, computer science and information theory and is considered a fundamental branch in the field of Artificial Intelligence.

Machine Learning is a recently trending field, although research and different approaches already originated in the 1940s and 1950s [4]. McCulloch and Pitts-neurons [58] forms the basis of neural networks today and originated in the 1940s. Samuel [80] researched on developing efficient computer programs for playing chess without explicit programming approaches as a method during the 1950s.

In recent years due to advanced development of algorithms and hardware, it has been possible to develop efficient algorithms. Many are good at learning complex tasks, otherwise impractical with explicit approaches. Neural networks, for instance, are very popular these days, and they have proven to be great models in multiple domains. In image classification and object recognition, with the breakthrough of *AlexNet* [52] in the ImageNet 2012 challenge [77], it was shown how deeper networks could improve the performance of the models, hence the trending name *Deep Learning* [30]. More recently, the same technology is one of the core elements behind the current development of autonomous vehicles [6, 54] and there are also numerous successful applications in other domains like language processing [61] and generation/translation tasks in language processing [14, 34, 35, 66]. In general, computer vision and language understanding may seem like trivial tasks for humans. However, developing generic computer algorithms with explicit programming that solves these tasks well is difficult.

Further on, we provide an overview of the main categories in ML. The methods in this thesis are concerned with supervised methods. We explain two of the two main categories, supervised learning and unsupervised learning. Although various categories like semi-supervised learning, reinforcement learning and feature learning [4] exist, they are considered out of the scope of this thesis.

2.2.1 Supervised Learning

Supervised learning methods [45] use labelled data and the goal of supervised algorithms is to determine a good approximation between a set of inputs and correct outputs. More formally, supervised methods approximates some function f that maps from some input space X to an output space Y , formally $f : X \rightarrow Y$. The iterative process of optimising this function to achieve a good approximation is known as the learning process.

An example is if we want an algorithm that classifies a specific object such as a ball. The input image has a corresponding output target that tells if the input image has a ball or not. By using labelled data, such an algorithm can iteratively be optimised to

learn particular features describing a ball. Hence, for unseen images, the algorithm can correctly determine whether or not an image contains a ball.

2.2.2 Unsupervised Learning

Unsupervised learning methods [45] does not use labelled data, and there is no explicit description of how an algorithm should optimise the understanding of observations. Unsupervised methods are concerned with finding patterns or groups in data, for instance, by clustering similar observations which seemingly belong in the same categories. For example, an unsupervised algorithm cannot tell if it is a ball or not. It may, however, group all similar images that potentially contains a ball in the same category.

2.2.3 Short on the curse of overfitting

Overfitting is a concept in statistical modelling and machine learning concerned with problems in which modelling a set of observations becomes too specific. That is, given a set of data points, any given model or function *overfits* the data if it draws too specific boundaries that are not general enough for the modelled problem. A simple example of overfitting is found in **Appendix A.1**.

2.2.4 Summary of machine learning

This section presented a general definition of machine learning and how the field has been trending in recent years. The development of advanced methods has resulted in algorithms that can solve complex tasks that usually are impractical with explicit approaches. We discussed how neural networks, one type of machine learning models, have successfully been applied in the field of image classification, language processing and various other areas, like self-driving cars and speech recognition and machine translation. Conclusively, we also discussed two different learning methods in the field, namely, supervised learning and unsupervised learning. The prior is concerned with the development of algorithms that use input-output pairs, to make a generalised and efficient approximation of the mapping from input to output. One example is neural networks. The second, unsupervised learning, does not use input-output pairs, and there are no explicit instructions on how to make a good approximation. Unsupervised methods try to find patterns and groups in data through clustering similar observations.

2.3 Neural Networks

A neural network, also referred to as a feedforward network is a supervised learning method that draws inspirations from how the human brain works [42, 74]. The "vanilla" neural network we know today is based on the Multilayer Perceptron (MLP), developed by Rosenblatt [74] in the 1950s. However, the MLP was developed as a

linear classifier that used threshold activation functions, which we will explain in detail in the following sections. On the other hand, neural networks as we know them today use non-linear activation functions [65, 68], which enables them to learn more complex patterns and decision boundaries. Nonetheless, the MLP is widely considered as the foundation for neural networks today.

A neural network f is a network of nodes that are computational models of the biological neuron. The most widely known model today is the McCulloch and Pitts-neuron [58] which originated in the 1940s, and motivated for the development of the MLP by Rosenblatt. **Figure 2.1** shows an example of a simple network, which has multiple layers with each layer encapsulating a set of nodes. Each node has connections between units in the preceding and succeeding layers and generates a signal, which is the output of a function and can be optimised, known as the learning process.

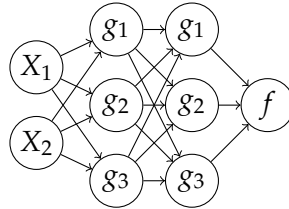


Figure 2.1: A simple neural network with 2 hidden layers. Note that we exclude the layer subscript for simplicity, hence the same subscript for the nodes in the hidden layers, which only indicates the j -th node.

2.3.1 Definition

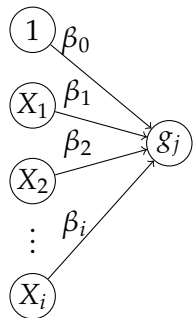
Consider a node g to be a computational model, a function with some inputs [58]. We can also understand this as an extension of the simple linear regression model [45]. It is a model concerned with predicting some output variable Y (the response variable), given some input variable X (the predictor), where the relationship between these two is assumed to be approximately linear. We can define the relationship as the following, where g represents the output variable Y , and is usually the conventional notation.

$$g \approx \beta_0 + \beta_1 X \quad (2.1)$$

Equation 2.1: The simple linear regression model.

β_0 and β_1 are the weights of the linear model, and is often interpreted as the intercept and slope for a linear fit to some observations [45]. In order to find the optimal approximation between X and Y , the weights of the model are iteratively optimised. This is usually referred to as the learning process, and we provide a more detailed overview in **Section 2.3.2**.

By definition, a neural network f is a collection of these nodes/functions, parameterised by all β in the network; $f(X; \beta)$ [33]. Each node g has multiple inputs rather than only one, where they are weighted independently. More formally, this is known as the multiple linear regression model [45], which is an extension of **Equation 2.1**. For any given node g_j , the model encapsulating multiple variables (regressors) X_i , is shown in **Equation 2.2**. Note that we refer to the i -th variable as X_i and a value of that variable as x_i .



$$\begin{aligned}
 g_j &\approx \beta_{0j} + \beta_{1j}X_1 + \beta_{2j}X_2 + \cdots + \beta_{ij}X_i \\
 &\approx \beta_{0j} + \sum_{i=1}^N \beta_{ij}X_i
 \end{aligned} \tag{2.2}$$

Equation 2.2: *The multiple linear regression model.*

2.3.2 The learning process

The learning process in a neural network can be considered as an iterative optimisation of the network f with a set of weights β , and is comprised of three steps; forward propagation, error computation and backward propagation. As an overview on a general level, we can understand this process as the following:

1. Propagating input signals x forward in the network to generate some output. The output is either a probability distribution P in classification problems or a continuous value/quantity \hat{y} in regression problems.
2. Measure how far off the predicted output or distribution is from the ground truth output/distribution with a loss function L to determine the loss.
3. Use the loss to adjust the weights β of the model in order to improve the function approximation.

In the following sections, we will refer to the prediction of a network as $\hat{y} = f(x; \beta)$, indicating that the predicted output is a function of the input value x for the given weights β .

Forward propagation

Forward propagation is the process of propagating the inputs forward in the network, generating some prediction \hat{y} . We earlier gave an overview of the general definition of a node g (**Equation 2.2**), which generates a weighted sum of the inputs. The forward propagating step generates a signal a (also referred to as the activation) as a function of node g , and is commonly known as the activation function. We can explain this in a general form, by denoting the activation function as ϕ :

$$a_j = \phi(g_j) \quad (2.3)$$

Equation 2.3: *The activation function for the j -th node, as a general scheme.*

The idea behind the activation is that neurons "fire", when the accumulated signal is above some threshold, similar to the chemical process in the neurons in the human brain. Rosenblatt [74] originally used the threshold function as defined in **Equation 2.4**. In recent years, however, there have been researched on many different activation functions that adds non-linearity to the networks [65, 68], in which they have been proven to be efficient for learning complex patterns. Examples of common activation functions as discussed by Nwankpa et al. [65] includes the sigmoid, tanh, ReLU, maxout and variants of these. Some of the commonly known activation functions are visualised in **Figure 2.2**.

$$a_j = \begin{cases} 1 & \text{if } g_j > 0 \\ 0 & \text{if } g_j \leq 0 \end{cases} \quad (2.4)$$

Equation 2.4: *The threshold activation of the j -th node.*

When predicting a quantity, a linear activation in the output layer can be applied, but for classification problems it is common to apply softmax [19, 52]. Softmax is also an activation function often applied on the output layer as a classifier in a network, because of its characteristics on providing class probabilities [19]. We will see in the following section how this enables us to compare probability distributions in classification problems.

Error computation

So far, we have looked at how a neural network generates predictions, either in the form of continuous values or probability distributions. Error computation is the step where the predicted output is compared against the ground truth, the labelled data.

The comparison can be done with a loss function L [45], and the result is often just referred to as the loss. The loss is used to determine how the weights should be adjusted to improve the approximation of the network $f : X \rightarrow Y$ with the backpropagation algorithm, which we will look at in the next section. Note that a common description for the loss between quantities in regression problems is also referred to as residuals [45] and there exist multiple approaches for comparing them. **Equation 2.5** shows the most common one, residual sum of squares (RSS).

$$RSS = \sum_i^N (y_i - \hat{y}_i)^2 \quad (2.5)$$

Equation 2.5: *The residual sum of squares [45].*

RSS is applicable to the simple linear regression model we defined in **Equation 2.1**. Variants of this loss function is also used, such as mean squared error (MSE) [45]

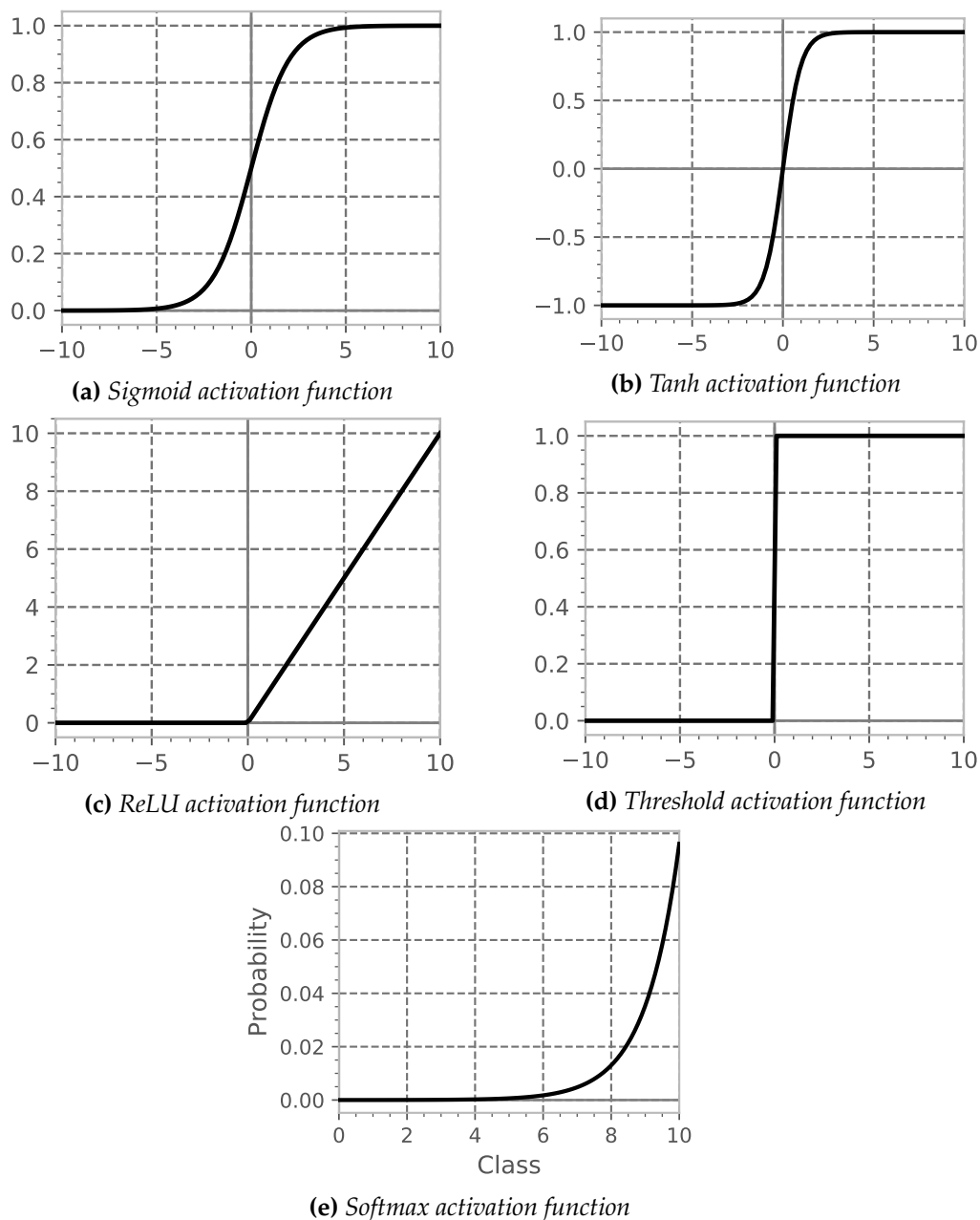


Figure 2.2: Commonly known activation functions used in neural network models. The Softmax activation function is often used in the output layer of a network.

which is the RSS average over all N samples. Nevertheless, RSS accumulates the differences in the predictions and ground truth values for all input samples. This gives us an estimate of how much the prediction deviates from the ground truth observations in quantitative measures, but is not ideal when comparing probability distributions in classification problems.

The common measure for comparing probability distributions [33] is the cross-entropy loss [45]. With this approach, we compare the predicted distribution P to the true distribution Q over all classes C . We formalise this in **Equation 2.6**, where i

denotes the i -th class.

$$L(P, Q) = - \sum_i^C P(\hat{y}_i) \log Q(y_i) \quad (2.6)$$

Equation 2.6: *The cross-entropy loss [45].*

From the equation we can see that there are some important characteristics with this loss function. First, the loss output is guaranteed to be non-negative, as James et al. [45] describes in the book *An introduction to statistical learning*. Second, a loss converging towards zero indicates increased prediction accuracy and decrease in prediction loss, implying better model performance of the network as a result of the model distribution P approximating the true distribution Q .

Backward propagation

We have now defined how we generate some predicted output and how we measure the prediction against the ground truth to find the loss. The learning process in neural networks is concerned with minimising this loss. We do this by adjusting the weights of the model slightly, with a delta, in the direction that minimises the loss. The backpropagation algorithm applies this rule iteratively, and is in essence an application of the gradient descent algorithm [33, 76]. An intuition of the backpropagation algorithm is described in **Algorithm 1**.

Algorithm 1 The backpropagation algorithm [76]

```

for each prediction  $\hat{y}$  do
  1. find the loss with loss function  $L$ 
  2. find the delta to determine how much to adjust the weights;  $\nabla L_\beta$ 
  3. propagate the deltas backwards for each layer in the network  $\nabla L_\beta$  and adjust
     the weights accordingly
end for

```

The common approaches for adjusting the weights are variants of the gradient descent method, which all take basis in the batch gradient descent method [33]. Before we explain these variants, we briefly describe how the general gradient descent, the weight update rule, works.

Gradient descent is concerned with minimising the loss function with respect to the weights of the model, by computing the gradients of the loss function¹. The loss optimisation of the total loss, often denoted as J , is seen in **Equation 2.7**. This equation defines the update rule of the weights of the network, where η is the learning rate and determines how much the weights are updated at each optimisation step.

Optimising the weights with batch gradient descent requires that the entire dataset is passed to the model as input for a given number of iterations. As Goodfellow, Bengio,

¹In general terms, gradient is a concept used in optimisation to define rate of change of any given function at any given point, with respect to the input. This is considered out of the scope of this thesis, but is mentioned because of strong presence in neural network optimisation.

$$\hat{\beta} = \beta - \eta \cdot \nabla J_{\beta} \quad \equiv \quad \beta - \eta \cdot \frac{\partial J}{\partial \beta} \quad (2.7)$$

Equation 2.7: *The batch gradient descent for updating the weights of a neural network, where $\hat{\beta}$ is the updated weight values, and β is the current weight values.*

and Courville [33] mention in the book *Deep Learning*, this method is deterministic because the gradients will be the same.

Stochastic gradient descent methods (SGD), on the other hand, are usually concerned with optimising the weights with multiple smaller batches of the entire dataset [33]. On the far left side of this scale, we can optimise the weights with one training sample at the time (on-line training). This means that the gradients will deviate more and contribute to more "noisy" optimisation because of the varying gradients from each sample [95].

On the other side of the stochastic gradient descent methods, we can train with larger fixed subsets of samples, known as mini-batches, hence the name mini-batch gradient descent. The size of these subsets is often referred to as the batch size. Compared to on-line training, we could think that larger batches are more preferred, which would result in less deviations. However, as Wilson and Martinez [95] discusses, on-line learning has also shown to outperform mini-batch gradient descent with larger batches. The batch size is thus a hyperparameter and choosing the right size may depend on the use case.

Nevertheless, the mentioned variants of the gradient descent method are often not used as is. Further extensions of gradient-based optimisation methods have been proposed in recent years, like Adam, AdaGrad and RMSprop [50, 75] and have more common applications. In general, they are more practical because of their adaptive nature on factors like the learning rate in addition to faster optimisations.

2.3.3 Summary of artificial neural networks

Neural networks are conceptual models that works well as optimisers. We have described how a node is modelled as a simple linear regression model (**Equation 2.1**), and how this can be extended for multiple nodes (**Equation 2.2**). By constructing a network of these nodes, a neural network can be defined as an optimisable function f that approximates some mapping between a set of input variables X and an output variable Y , where the optimisation is known as the learning process. The learning process is achieved by propagating some signals forward, generating some predictions, comparing the predictions with some loss function and then backpropagating gradients of the loss function to update the network weights.

2.4 Recurrent neural networks

In this section, we give a theoretical overview of recurrent neural networks (RNNs), one type of neural networks designed for sequential modelling. First we present

formal definitions, followed by an overview of the learning process. Further on, two RNN architectures that have common applications today are discussed, followed by a summary in the end.

2.4.1 Definition

Consider a simple neural network as described in **Section 2.3**, but where nodes in each layer now has in-between connections (recurrent connections). The result is a recurrent neural network (RNN) as shown in **Figure 2.3**, where the connections represent temporal dependencies (dependencies in time) and introduce an extra set of optimisable weights. RNNs are another family of neural networks designed for applications on sequential problems like language processing [14, 35, 79] and the most simple RNNs only introduces these recurrent connections.

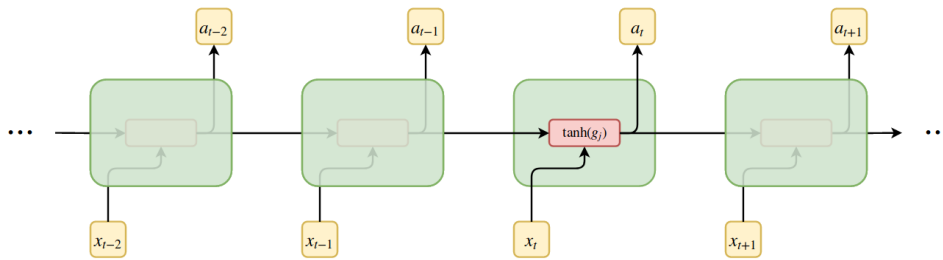


Figure 2.3: An illustration of 4 standard recurrent units in a layer with their respective recurrent connections. The output signal is a function of the current input and recurrent input at time t and $t - 1$, respectively.

Formally, as shown in **Equation 2.8**, we let the input at time t be defined as x_t and recurrent inputs from the previous timestep be defined as a_{t-1} , both being vectorised inputs. Similar to the weighted sum we described earlier, the latter denotes the signals of the weighted sum g_{t-1} of the previous recurrent unit. By introducing recurrent inputs, we introduce a new set of weights. We let the input weight set be defined as β_{ih} and the recurrent input weight set be defined as β_{hh} , respectively denoting input-hidden and hidden-hidden weights.

$$\begin{aligned} g_t &= \beta_0 + [\beta_{ih} \cdot x_t] + [\beta_{hh} \cdot a_{t-1}] \\ a_{t-1} &= \tanh(g_{t-1}) \end{aligned} \quad (2.8)$$

Equation 2.8: Model of the recurrent unit in simple recurrent networks (RNNs). The input signal is a function of the input at time t and time $t - 1$, forming two sets of optimisable parameters where \cdot denotes the dot-product. The recurrent input is the previously generated output signal as a result of the \tanh -activation.

2.4.2 The learning process

The learning process in recurrent networks is similar to earlier definitions for feedforward networks. Signals propagate forward through the network and tanh is usually applied for non-linear activation of the weighted sum, hence $a_t = \tanh(g_t)$ being its specific form. Lastly, because RNN units are recurrent applications of themselves, the backpropagation of gradients is an ordered operation through each timestep. This is an extension of the ordinary backpropagation algorithm known as backpropagation-through-time (BPTT).

2.4.3 The difficulty of learning

We have now described the mechanism around vanilla RNNs. This is an architecture designed for solving sequential problems, as ordinary neural networks does not take temporal dependencies into consideration. Nonetheless, a common problem in these networks is the vanishing- and exploding gradient problem [8, 40, 67] which makes it difficult for the networks to capture temporal dependencies over longer timespans. Although this is a widely known problem for vanilla RNNs that use gradient-based optimisation techniques, alternative approaches have been proposed [8, 57, 67] to overcome the problem. The most notable contribution however, is the introduction of gating mechanisms in RNNs, giving rise to new architectures like Long Short-Term Memory units (LSTM) and Gated Recurrent Units (GRU).

Long Short-Term Memory units

Long Short-Term Memory (LSTM)-based networks [41] use carefully designed LSTM-cells. These are computational units in networks controlling information flow through gating mechanisms. Initially proposed by Hochreiter and Schmidhuber [41], the architecture was designed to overcome the above-mentioned problem of learning long-term dependencies. One LSTM unit is composed of multiple gating mechanisms, namely, the forget gate, output gate and update gate. Collectively, they enable a possibility to maintain the overall cell state in each unit such that a LSTM-layer is able to effectively capture longer temporal dependencies.

As the names imply, the subset of information to be removed from the cell state is determined by the forget gate and the new information to add to the cell state is controlled by the update gate. The output gate is concerned with controlling the output flow from the cell state, and thus determines what information is passed onwards to the succeeding LSTM-unit.

Gated Recurrent Units

The Gated Recurrent Unit (GRU) [14] is a relatively new contribution compared to the LSTM-units introduced in 1997, and is quite similar. They were initially proposed by Cho et al. [14] and have also shown to be at least as good as LSTM-based networks in recent years [16, 46]. The popularity around the GRU-architecture is primarily

because of their less complex structure. It differs from LSTM by reducing the overall complexity of the unit. In general, it does not include an output gate that controls the output flow from the cell state. By doing this, the overall number of optimisable parameters is reduced, and hence the model complexity is lowered.

2.4.4 Summary of recurrent neural networks

Recurrent neural networks (RNNs) is a specific type of neural network architecture designed for solving sequential problems. The core principle is introducing recurrent connections between nodes in each layer, which makes it possible to model temporal dependencies. We explained the vanilla RNN unit and how it is a function of multiple inputs, the current input at timestep t and the recurrent input from $t - 1$. This architecture works well for sequential modelling, but one shortcoming is the learning problem in RNNs, making it difficult to learn dependencies over longer timespans. Two architectures that overcome this problem is the LSTM-unit and GRU, both introducing gating mechanisms which enables recurrent networks to learn long-term dependencies as well.

2.5 Convolutional neural networks

We give a theoretical overview of convolutional neural networks (CNNs). First, we look at formal definitions and the learning process, in which the most important components in CNNs are described. We describe convolutions and pooling, before giving a summary in the end.

2.5.1 Definition

Convolutional neural networks (CNNs) are a family of neural networks that are commonly applicable to problem domains where spatial information and other grid-like topologies are of importance [33, 55]. The most common application is image classification [52], as images can be considered 2D-grids. However, CNNs are also applicable on time-series [66], which in essence are 1D grids with samples at fixed time intervals. Overall, the main difference between standard feedforward networks and CNNs, is the architecture.

Consider a standard feedforward neural network which we described in **Section 2.3** about neural networks. There are two concerns when applying a feedforward network to grid-like topologies with more than one dimension. The first is that the spatial information is not preserved, as the grid has to be flattened to the networks. Secondly, an application like this does not scale for use cases like image classification and other grid-like topologies, due to the increased number of parameters as a result of many interconnected neurons on the grid. An increase like this would require more computational resources, as the optimisation problem require more effort.

CNNs solves these problems by only looking at a subset of the input, known as local connectivity. The number of weights are reduced and spatial information is

still preserved by applying two additional mechanisms; the convolution- and pooling operation. We further explain these in detail in the following section on the learning process in CNNs.

2.5.2 The learning process

Similar to what we described earlier in **Section 2.3.2** about learning in feedforward networks, inputs propagate forward in the network and generate some predictions. Backpropagation and forward propagation of signals is conceptually equal to vanilla neural networks. However, the difference in CNNs exists in their architecture, where convolutional- and pooling layers are introduced. The general scheme of a CNN is shown in **Figure 2.4**.

Convolutional layers

Convolutional layers in a CNN take advantage of the convolution operation, a mathematical operation on two functions, which generates a third function explaining an estimated relationship between both [33]. In the context of grid-like topologies, the convolution operation can be considered as a sliding window over the grid input, usually referred to as convolving a filter or kernel on the input. **Figure 2.4** illustrates the convolution operation in CNNs.

Each convolutional layer contains a set of learnable filters, more specifically a set of weight matrices. The number of learnable filters is equivalent to the number of nodes in a layer, as each node represents a filter learning different features. The filters are spatially small and are convolved on the input by computing a weighted sum to generate an output volume, the set of feature maps. Because one feature map is a linear activation, each map is passed through a non-linear activation function, similar to the process in feedforward vanilla networks.

However, the output volume of a convolution layer is not fixed. The process of adjusting the output volume is dependent on multiple factors, introducing some additional hyperparameters. For instance, step size (stride), filter bank (number of filters), filter size, padding and dilation rate are additional mechanisms in the convolution layer that determine depth and spatial dimensions of the output volume. We will not describe these in detail in this thesis, but their importance in the process of model optimisation should not be disregarded.

Tuning these hyperparameters share one common motivation, which is to reduce the model complexity by reducing the number of parameters and increase local connectivity. This enables filters to look at larger proportions of input, without introducing additional computational complexity.

Pooling layers

A pooling layer in a CNN is an additional mechanism controlling spatial dimensions and reduces model complexity and the number of optimisable weights. They are

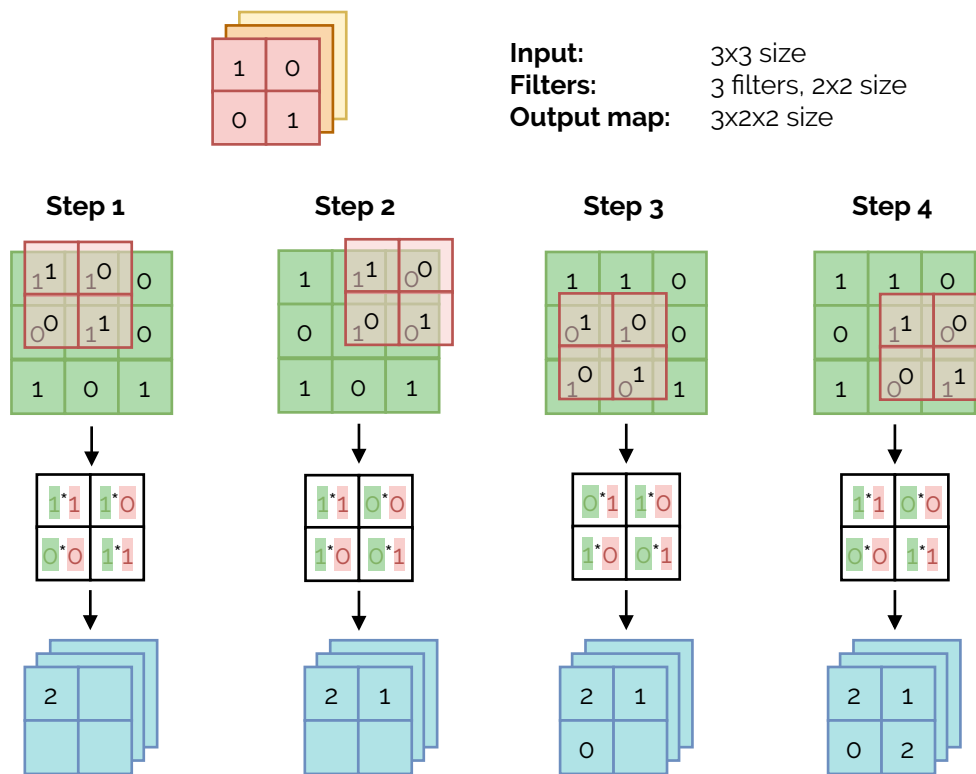


Figure 2.4: Illustration of the convolution operation in CNNs for 2D-grids. The operation is performed for each filter on the input, resulting in an output volume with the same spatial size as the number of filters and the filter size. Three 2x2-filters, thus results in an output dimension of 3x2x2.

more common to use for downsampling activations to lower spatial dimensions, but pooling layers are also applicable for upsampling in networks, for instance in encoder-decoder architectures.

As shown in **Figure 2.5**, two different pooling techniques are max pooling and average pooling. The names are as intuitive as they imply. By dividing the input to the pooling layer into multiple smaller sub-grids, each sub-grid represents a portion of the input volume. Max pooling is a technique indicating that the maximum value in a sub-grid is the representative pixel value for that portion of the input volume. Average pooling computes the average of each sub-grid, yielding a smoother representation.

2.5.3 Summary of convolutional neural networks

Convolutional neural networks (CNNs) are derivations of the standard feedforward neural network described in **Section 2.3**. They have common applications in domains where spatial information is of importance, or in general where data emits certain grid-like topologies like images (2D-grids) or time-series (1D-grids). We have described why vanilla neural networks are impractical for these types of problems. In essence, the computational complexity increases as a result of many interconnected

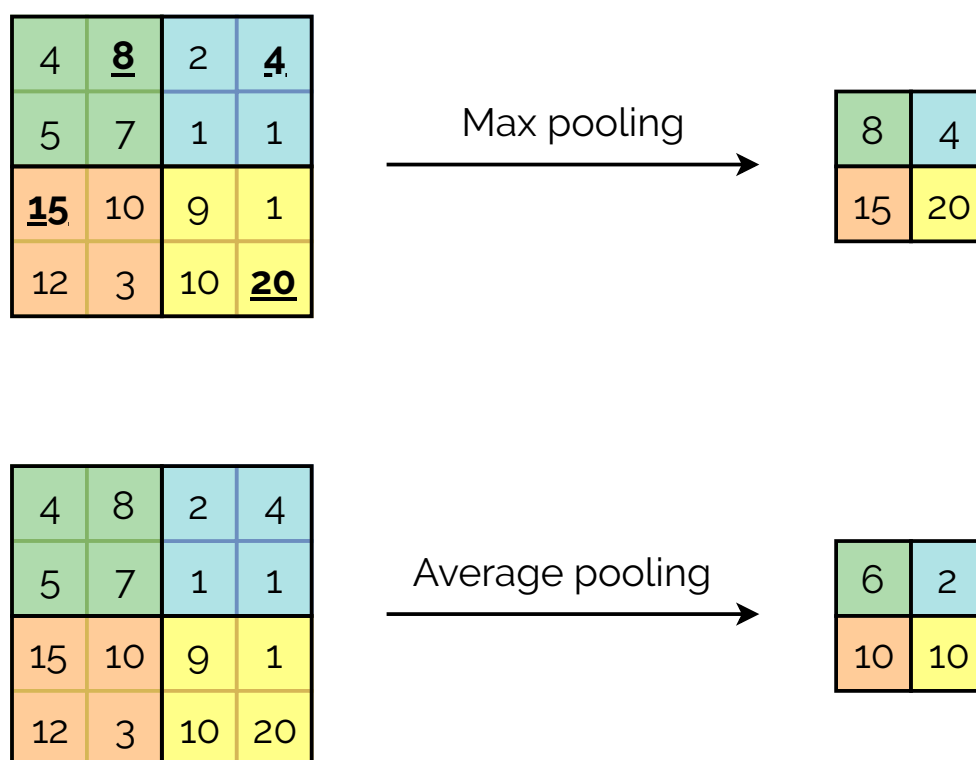


Figure 2.5: Illustration of 2x2 max pooling and average pooling. As the names imply, the 2x2 sub-grid of the input downsamples to either the maximum value in the grid or the average value.

nodes and the spatial information is not preserved. To avoid this problem, CNNs introduces the convolution- and pooling mechanism, which distinguishes them from the vanilla neural networks. These are mechanisms that looks at portions of the input and thus preserves spatial information and reduces overall complexity with localised connections.

2.6 Summary of background

In this section, we first discussed related work. First, we looked at the work by Bai, Kolter, and Koltun [5], who compared CNNs and RNNs on various language tasks. Although their comparative study presents a good baseline for language tasks, the application in the time-series classification domain is yet to be explored. We then discussed the work by Wang, Yan, and Oates [93], who present a stronger baseline for how neural networks are applicable to this domain. In their paper, they particularly showed how CNN/FCN outperforms traditional time-series classification models across 44 datasets. Moreover, a similar study was performed by Fawaz et al. [26], who presented a more comparative study on various deep learning architectures as well, which is the most extensive to date. However, the latter only explored one type of recurrent networks, namely, Echo State Networks. Although Fawaz et al. [26] show that their compared CNNs overall seemed to be more efficient, it is uncertain

how the application of LSTMs, a state-of-the-art architecture, compares in a similar study. Although the last paper by Siami-Namini and Namin [83] explores LSTMs in the time-series classification domain with canonical results, the comparison focused on ARIMA, a traditional statistical model used in time-series analysis.

In the following sections, we first outlined definitions in the field of Machine Learning and discussed supervised and unsupervised learning. Further on, we discussed the theoretical aspect of neural networks. We first presented Artificial Neural Networks (ANNs), the vanilla neural network models. We looked at formal definitions, described the learning process and presented a summary.

We then presented the two main architectures, Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), following the same scheme by providing definitions, describing the learning process and at last, a summary. For the RNN, we explained how they differ from vanilla networks through an additional parameter between nodes in each layer, modelling temporal dependencies. We also described various derived architectures like LSTM and GRU, which are designed to overcome a general problem in vanilla RNNs of capturing long-term temporal dependencies. For CNNs, on the other hand, we described how they are designed to preserve the spatial structure of the input. We explained the two main mechanisms in this regard, the convolution operation and pooling operation. Both enable (i) localised connections that look at portions of the input and thus require less computational effort and (ii) still preserves spatial information.

Nevertheless, we have seen that LSTM, RNNs in general and CNNs have many applications areas, in which they have proven to be efficient in various tasks, especially in language processing. Their applications have also been notable in the time-series classification domain in recent years. Evidently, there are limited comparative studies of RNNs and CNNs in general, in the time-series classification domain. Therefore, this thesis aims to provide such a comparison using various case studies, and in the following chapter, we present our methodology.

Chapter 3

Methodology

To evaluate LSTM and CNN and provide a comparison, we develop a system testbed to perform experiments and analyse results. Recall from the related work in the previous chapter, where we observed that there are limited comparisons on RNNs and CNNs on sequential problems, and even less in the time-series domain. While these architectures have been applied separately or used in hybrid approaches on various use cases, relatively few studies present an empirical comparison of both. This further, highlights the importance of our research question in **Section 1.2**. In this chapter, we present the methodological approach in answering this question. We describe design decisions in terms of model development and hyperparameter selection and the process from data analysis to experiment execution.

3.1 Overview

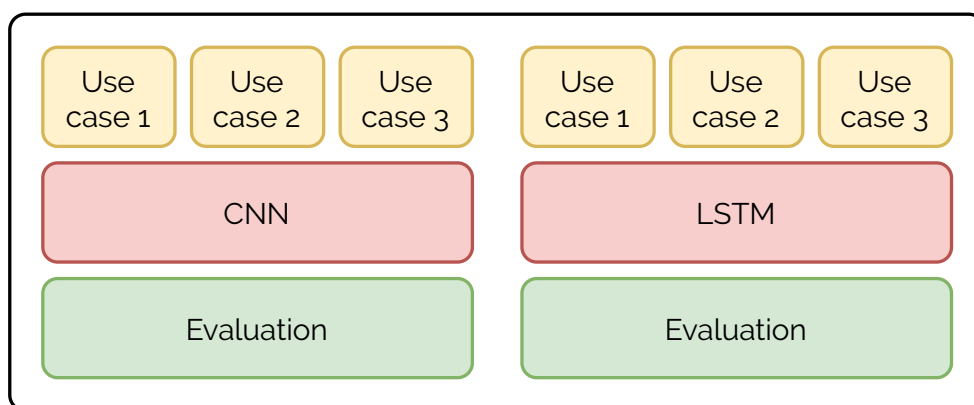


Figure 3.1: General-level overview of the methodology. Each use case is tested separately for each model and evaluated accordingly. First, we select a use case, perform data analysis and run initial experiments for a given model with corresponding evaluation. Further on, we perform manual hyperparameter optimisation, run experiments for model optimisation and evaluate the results correspondingly.

Figure 3.1 illustrates the general overview of the method of this thesis, which

highlights how CNN and LSTM are evaluated across the following three tasks:

- U_1 : **Depression detection** - In **Section 4.1**, we compare CNN and LSTM on a binary classification problem with a publicly available dataset to classify depressed patients based on motor activity levels.
- U_2 : **Energy prediction** - In **Section 4.2**, we compare CNN and LSTM on a multiclass classification problem with a dataset collected from Electric Vehicles (EVs) the past year to classify energy demands for EVs.
- U_3 : **Football readiness classification** - In **Section 4.3**, we compare CNN and LSTM on a multilabel classification problem with a dataset from an existing performance monitoring system called PMSys to classify readiness of football players.

We explore the mentioned datasets use case by use case, comparing both architectures. Vanilla RNN is not used, because as we have described, gated architectures have shown to be more promising in recent years. Moreover, we exclude the GRU architecture in this comparison as well because of two reasons. The first is related to time and resource constraints and the second is because we aim to present a thorough comparison of one gated architecture. We look at both LSTM and CNN in addition to the performance of CNNs in the time-series classification domain. Ideally, a comparison of CNN against both GRU and LSTM would be preferable, similar to the work presented by Bai, Kolter, and Koltun [5], to provide better empirical evaluations.

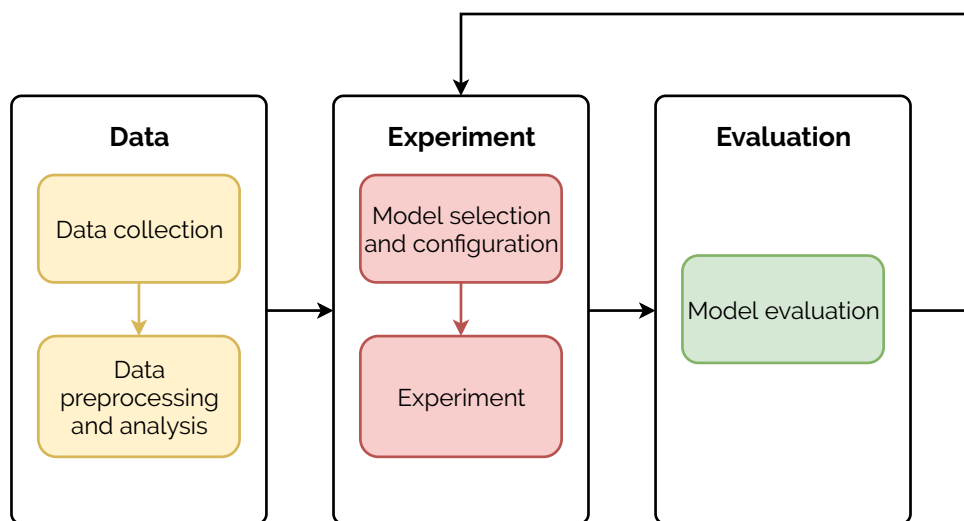


Figure 3.2: Detailed flow of the overall methodological process. The data-part of the methodology relates to collection, analysis and preprocessing. The experiment part explains how we select models, set configurations and hyperparameters and run experiments. The evaluation part relates to how we evaluate each run and adjust initial hypotheses formed during the data analysis, motivating for other hyperparameter configurations.

Moreover, **Figure 3.2** illustrates a more detailed overview of the flow in our methodology, which separates into three steps. The data-related step concerns with the collection of data, the preprocessing and data analysis. Through the data analysis, we form initial hypotheses, and with a combination of earlier best practice, the initial

hyperparameters are chosen. In the experiment step, we select a model and use these initial configurations. Lastly, we evaluate the results, explore the hypotheses further based on the model performance and tune the hyperparameters accordingly.

3.1.1 Data preparation and data analysis

As described above, we evaluate the models across three different tasks. The first use case, U_1 , is a univariate time-series whereas U_2 and U_3 are multivariate time-series¹.

For each dataset, we perform explorative data analysis and visualisations to understand the underlying data distribution, topology and temporal dependencies. One standard method we apply to all use cases to understand temporal patterns and underlying time structure is the autocorrelation function (ACF). The result of this analysis is also used to determine initial input sequences for the models.

ACF is a function that calculates a correlation coefficient of a set of observations and a copy of itself [82] with n delayed timesteps. This delay factor is also known as *lag*, and the coefficient is a value between -1 and 1, where -1 indicate a negative correlation and 1 indicates a positive correlation. For instance, if the coefficient is higher for smaller lags, it indicates that there is a positive correlation between a given observation x_t at time t and a recent observation at x_{t-n} .

Apart from ACF, we also analyse the data through visualisations, like heatmaps and line charts. The overall aim is to understand the underlying data distributions and form initial hypotheses that prove as a basis for initial hyperparameter configurations.

3.1.2 Initial experiments and optimised experiments

Recall from **Figure 1.3** about the general-level experiment design, where we illustrated the flow in our experiments. As we described above, we form some initial hypotheses which motivate for how we configure our models through initial hyperparameter configurations. Overall, the model optimisation is separate from the initial experiments, only differentiated by which hyperparameters we choose to explore in detail for optimisation. General observations made in the initial hyperparameters motivate for how we choose which ones to optimise further. For instance, on some occasions, the models showed bad convergence, in which we tried various configurations on learning rate and momentum to explore how the models could be optimised well.

3.2 System specifications

The Python programming language [71] is used to perform data analysis, develop models, run experiments and evaluations. With a rich ecosystem and a diverse set

¹Time-series with one time-dependent variable is considered univariate and multivariate series are series with multiple time-dependent variables. For instance, EV energy demand may not only depend on historical observations but other factors such as temperature and driven distances, thus being multivariate.

of supported libraries and frameworks, tasks related to data analysis and neural network modelling are more convenient. Further on, **Table 3.1** summarises what software, hardware and frameworks that are used.

Software		
Name	Version	Description
Ubuntu Bionic Beaver (LTS)	18.04.2	Operating System
Python	3.6.7	Used for implementation
Keras	2.2.4	Used for building models
Pandas	0.23.4	Used for data analysis
Tensorflow	1.12.0	Used as backend for Keras
CUDA	9.0.176	Required for Tensorflow
cuDNN	7.4.1	Required for Tensorflow
Hardware		
Name		Description
CPU		Intel i7-2600
GPU		NVIDIA GeForce GTX 980
Memory		Kingston 8 GB DDR3
GPU Memory		4 GB, GDDR5

Table 3.1: System specifications of hardware and software.

Pandas [59] and Keras [15] have most direct applications in this thesis. The former is extensively used in the field of data analysis and especially for time-series analysis. Keras however, is a high-level framework for developing neural network models. It is designed to be built on top of existing platforms like Tensorflow [1], which is used as the backend for Keras and facilitates for computational efficiency. In this thesis, the GPU-release of Keras is used, enabling GPU-optimised computations. The corresponding drivers and interfaces responsible for the underlying interaction between Tensorflow and hardware require additional installation and configuration. Namely, CUDA [62] and cuDNN [13] are additional underlying requirements when installing Keras with Tensorflow.

3.2.1 Tensorflow

Tensorflow [1, 90] is an open-source framework designed for large scale numerical computations. It has particular support for machine learning and deep neural networks and is supported on most platforms. Additionally, there is an extensive open-source community actively engaging in the development of Tensorflow [89].

In general, Tensorflow represents computations in a computational graph, where nodes represent operations and edges represents tensors transitioning from one state of the graph to another. A tensor is a multi-dimensional array flowing in-between each operation. This model of computing is referred to as the dataflow paradigm, where information is a functional transformation of operations.

In terms of architecture, Tensorflow is implemented as a layered architecture.

Applications and libraries like Keras utilise underlying computations, hardware interactions and optimisations implemented in the Tensorflow core/kernel as an application on top. For instance, implementations in Python and C++ are initial implementations whereas other examples include JavaScript, Java and Go support. A detailed overview of the architecture is discussed in the original paper by Abadi et al. [1].

Further on, because Tensorflow is built for optimised numerical computations, it is thus important to describe the core elements like CUDA and cuDNN, required for the GPU distribution. In this thesis, we utilise the underlying performance gain of using GPUs for computations. Because Tensorflow has a specific release tailored towards GPU-applications, it uses CUDA [62] and cuDNN [13] to achieve this. They are both interfaces to Nvidia GPUs, tailored for deep learning and being a general interface, respectively. Although being implied requirements for Tensorflow, the libraries are not discussed in detail in this thesis.

3.2.2 Keras

Keras [15] is a machine learning library primarily designed for neural network modelling. It is also open-source and has gained popularity in recent years. Keras has a simple and intuitive interface for the development of neural network models. An example of the application of Keras is found in **Appendix C.1**. Moreover, the library can be run on top of multiple machine learning platforms like Tensorflow and Theano and reduces the threshold of complexity when developing neural network models. Overall, the framework works well as a high-level application interface.

Keras has a modular implementation of core elements in neural network models, which includes different optimisers, layers and layer types, activation functions, metrics and regularisers. Keras also includes pre-trained models, many of the models being implementations of earlier ImageNet contributions, which can be applied easily. Examples include more complicated classification and prediction tasks that require such models or transfer learning tasks [11, 39].

In general, Keras functions as an abstraction layer on top of the more technical platforms like Tensorflow. The application interfaces are consistent and intuitive, which provide modular functionality. For instance, implementations of the more complex gating units in recurrent networks, like LSTM-units, are applied in this thesis. Such modules are also easily extensible, facilitating for custom implementations as well. In this thesis, however, custom units and implementations were not required, and thus not used.

3.2.3 Pandas

Pandas [59] is a library used for data analysis. It is open-source and provides a good interface for advanced analytics and data manipulations. The library is tailored well for time-series analysis in particular. For instance, Pandas has an interface for grouping and aggregating observations, upsampling/downsampling of a time-series

and well-defined interfaces applicable to data interpolation and cleaning. A simple example of the use of Pandas illustrated in **Appendix C.2**.

Moreover, Pandas takes advantage of structured data, tabular datasets and vectorised operations and represents the data in Pandas data frames or Pandas series. Each data type has its own set of data manipulation operations. For time-series data, Pandas is also able to automatically infer and parse timestamps and creating an index so that operations on a time-series becomes a less tedious task. While Python provides similar parsing features, this functionality is highly preferable and less time-consuming.

Many of these features are used extensively in this thesis, especially for data analysis. They lower the complexity of exploratory data analysis, making it easier to derive descriptive statistics and intuitive visualisations. We aggregate, filter, clean and resample the time-series with Pandas, which would otherwise become a manual task. Additionally, Pandas is used for generating descriptive statistics and figures through the built-in plotting interface²

3.2.4 Experiment framework

Figure 3.3 shows the overall file-structure of the implemented experiment framework, which we divide into three steps. The first step is concerned with setting configurations in a JSON-file named `config.json` for each experiment. Second, we run the implemented experiment pipeline with a Python-script named `run.py`, which loads the configurations and sets up the experiment accordingly. Both these files can be found in **Appendix C.3** and **Appendix C.4**, respectively.

```
{
  "_usecase": "depression",
  "_experiment_folder": "exp_depression",
  "network": {
    "architecture": "cnn",
  }
}
```

Listing 3.1: *Smaller subset of config.json where use case, experiment folder and network architecture is selected. Upon execution, the configurations are loaded, and models are built and trained.*

```
$ (thesis) python3 run.py
```

Listing 3.2: *Example of how to execute experiments after configurations are set in config.json. run.py runs the experiment and evaluation pipeline automatically.*

Overall, as illustrated in **Listing 3.1** and **Listing 3.2**, the described process runs automatically after configurations are set. Our pipeline loads the JSON-configurations through `config.py` and builds and trains the models through `model.py`. `history.py` tracks the training- and validation history across cross-validations. The last step is saving

²The plotting interface for Pandas is built on top of Matplotlib, a Python library used for creating figures. Matplotlib is used in the thesis for generating figures, but is not explained in detail here.

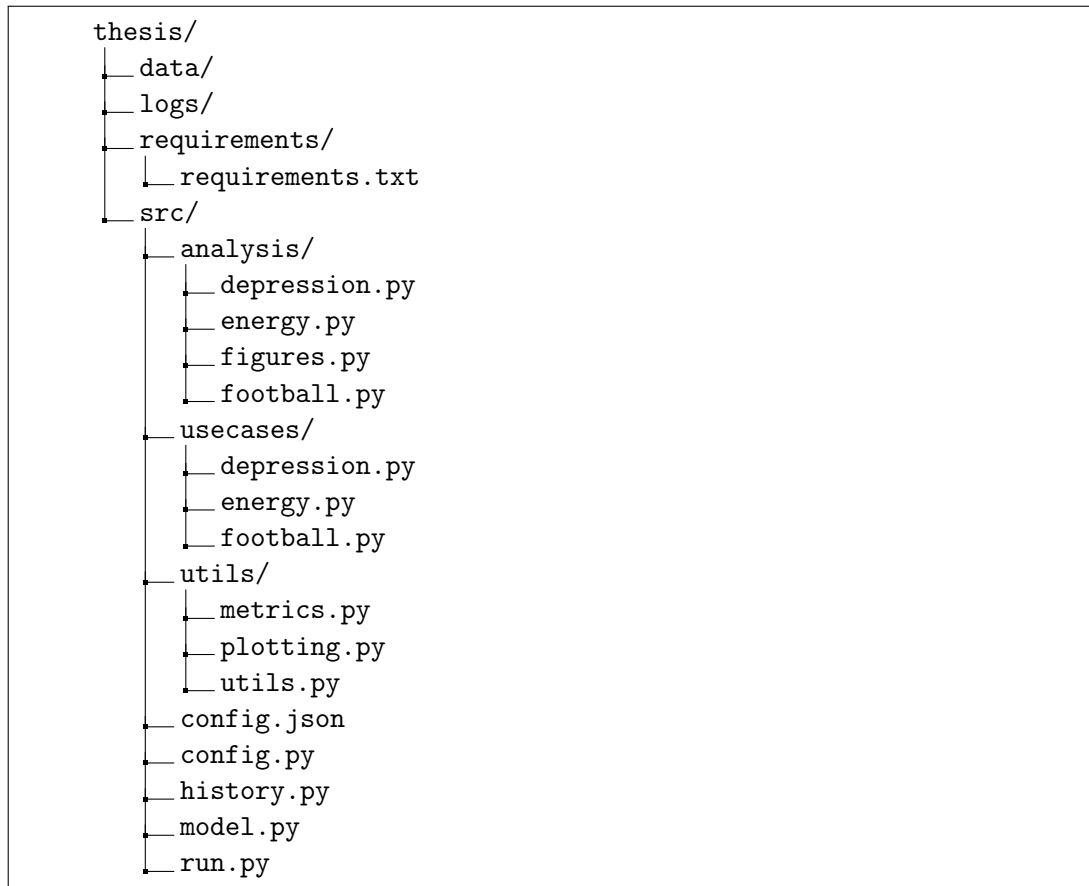


Figure 3.3: Tree structure of implemented experiment framework.

evaluations and experiment results to persistent storage, which we describe in more detail further down.

Figure 3.4 gives an in-detail overview of our proposed framework from use case selection and data analysis to experiment pipeline and evaluation pipeline. Each module in our pipeline is generic and can be extended to include other architecture types or custom implementations. In order to extend this system, like adding data-formatting scripts related to specific use cases, we can include this in the `usecases-` folder and integrate it accordingly. Overall, the tree structure observed in **Figure 3.3** illustrates the in-detail overview of the proposed framework.

For the evaluation pipeline, however, the tree structure is seen in **Figure 3.5**. As seen, we save most experiment-related information to persistent storage, most importantly the configuration file. We store the raw matrices and generated heatmaps, in addition to various other outputs like training- and validation history, related figures, models, history statistics and training times. Overall, this output is standardised and created based on the settings in the configuration file, where an output folder for a given experiment can be specified as well.

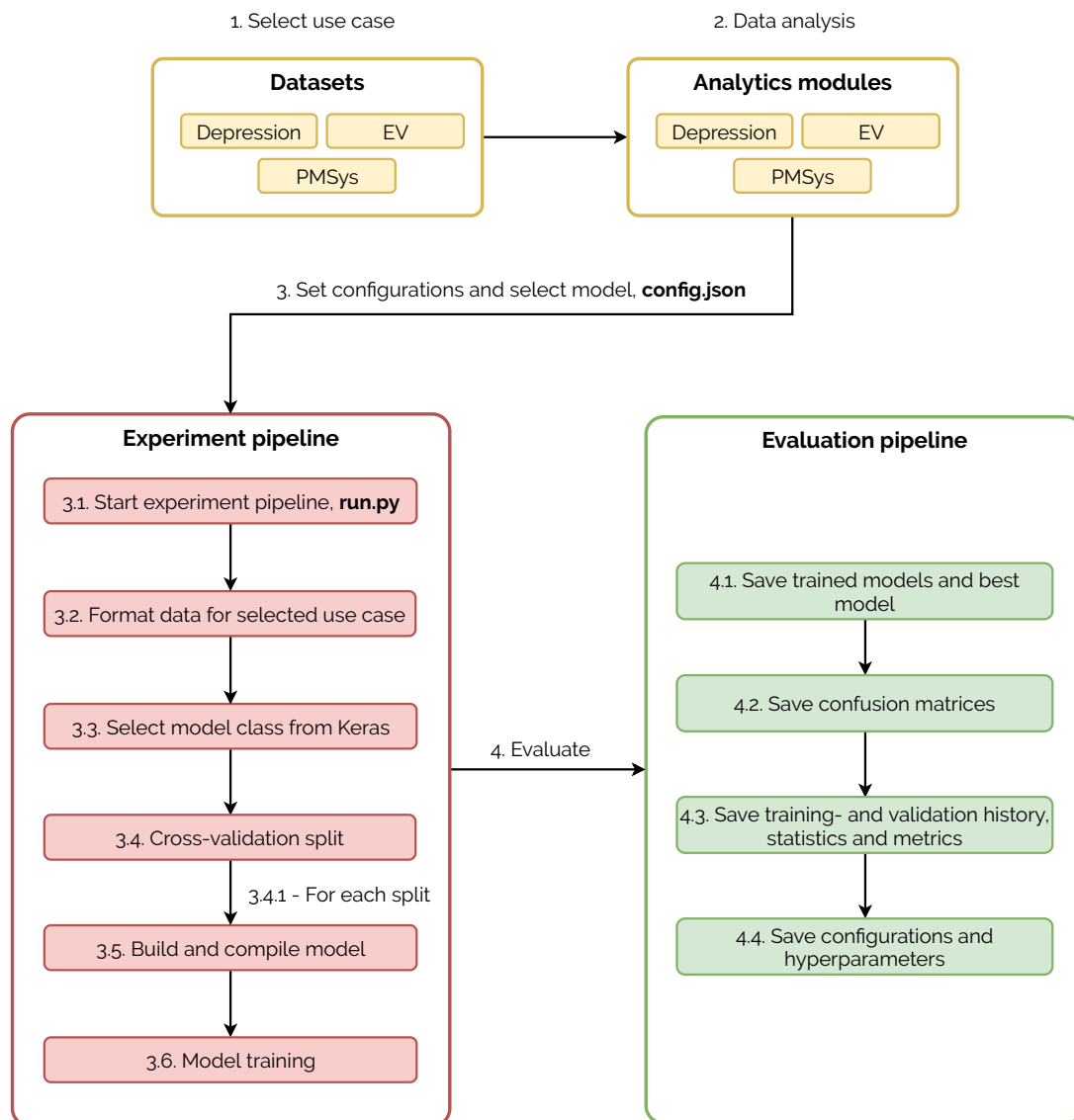


Figure 3.4: Overview of developed experiment framework and pipeline from data analysis through experiment execution and evaluation. Each color can be attributed to the figure defined in the first section of the general overview.

3.3 U_1 : Depression detection

This section gives an overview of the first use case. We present a general level background, the dataset and the data analysis which provides motivations on initial hypotheses that are explored.



Figure 3.5: General tree structure of stored results and generated logs from the experiment- and evaluation pipeline in our experiment framework. Confusion matrices are stored, accordingly with their respective heatmaps. Our evaluation pipeline also saves the models, best models, history statistics, history, configuration file and history figures.

3.3.1 Medical background

Depression

Depression is a common mental disorder which has a broad definition. In general it is considered as a mood disorder and is characterised by reduced pleasure (or low mood) or in most day-to-day activities like sleeping, eating and working [22, 63]. Common symptoms include the feeling of hopelessness or guilt, decreased energy levels and in more extreme cases, suicidal thoughts [63]. As reported by the World Health Organisation [23], close to 800 000 people die due to suicide every year as a result of depressive disorders. The experience and severity of the symptoms vary, as some experience every symptom and others experience only a few. It is thus important to understand the causes of a depressed patient, and the severity of it, in order to treat patients appropriately.

As mentioned by The National Institute for Health and Care Excellence (NICE) [22], the severity of depression, which can range from mild to severe is related to multiple factors. Some factors include number of symptoms, their seriousness, duration and how it affects day-to-day activities and general functional aspects of daily life. With 4-5 vary symptoms over 2 weeks with higher severity, the general agreed diagnostic systems diagnoses this as major depression.

Various degrees and forms of depression can have multiple side-effects and causes in general. More often, causes include genetic, biological, environmental and psychological factors [22].

Forms of depression

As mentioned, depression has a broad definition. We described how the severity of depression can vary but there are also multiple forms of depression. To mention some few, this includes seasonal affective disorder, postpartum depression and unipolar/bipolar depression [63], the latter being the form of depression included in the depression dataset we describe in the next section.

Compared to unipolar depression, which is the common form of depression with diagnosed symptoms for at least two weeks [63], bipolar depression is a form of depression where there are unusual and intense shifts in energy, activity levels and day-to-day activities. A bipolar depressed patient experience periods of depression or mania (periods of extreme excitement), where the latter does not occur in unipolar depressed patients [64].

Within the bipolar depression spectrum there are four basic types, where we will focus on bipolar 1 and bipolar 2 depression, as these are the main disorders included in the depression dataset in addition to unipolar depressed patients. The difference in both bipolar types is that the prior include more intense shifts with longer periods of depressed and manic episodes, or in some cases, mixed episodes [64].

Depression detection and machine learning

Because depression is a broad spectrum and the severity of depression varies from mild to severe, it is important to diagnose patients appropriately. A significant motivation is thus how data-driven models can contribute and provide a better basis for medical staff to provide appropriate and effective diagnosis of depressed patients.

Medical data, and especially data related to depression is not common. Recent research on detecting depression have been proposed by using audio and text data [3]. The study shows how sequential models like LSTM-networks can be used, where responses from patients undergoing depression screening was analysed.

However, there are multiple causes and severities of depression, one of them potentially being physiological, like motor activity. Garcia-Ceja et al. [29] present a dataset on motor activity levels collected from depressed patients, where they apply multiple machine learning models to provide a baseline for further research and potential improvement. The overall findings suggests that the measured sensor data can be used to detect depression.

3.3.2 Dataset

Overview

The *depression* dataset [91], further referred to as the depression dataset, consists of data collected from 23 conditioned/depressed patients and 32 control patients. It includes minutely motor activity recordings acquired with sensor readings from an actigraph watch³, and is originally collected for study of motor activity in schizophrenia and major depression [10].

	Patients	Average	SD	Min	Max	Total
Condition	23	12.65	2.77	5	18	291
Control	32	12.56	2.31	8	20	402

Table 3.2: Overview of depression dataset [29]. The table shows the total number of patients, and statistics on number of days. We can see in the table that the total number of days collected is higher for the control group, resulting in a class imbalance in the dataset.

The actigraph watch measures activity levels by registering movements above 0.05g. Movement above the threshold is considered as activity, further accumulated as an activity count [29]. A higher count is thus proportional with the activity intensity, representing higher activity levels. In the following section, data analysis is carried out for both groups, in which the motivation is to form initial hypotheses and design of experiments.

³An actigraph is a watch monitoring physical activity

Data analysis

The activity counts for both groups are compared to provide an overview of the activity levels and intensity in general. **Figure 3.6** shows the average level intensity and a heatmap of the average activity levels during night for both groups. Notably, the overall intensity on a hourly basis is lower for the depressed group, whereas the intensity in activity is more consistent for the control group during the evenings and mornings. Additionally, the overall sleeping pattern during the night indicates irregularities among the depressed patients.

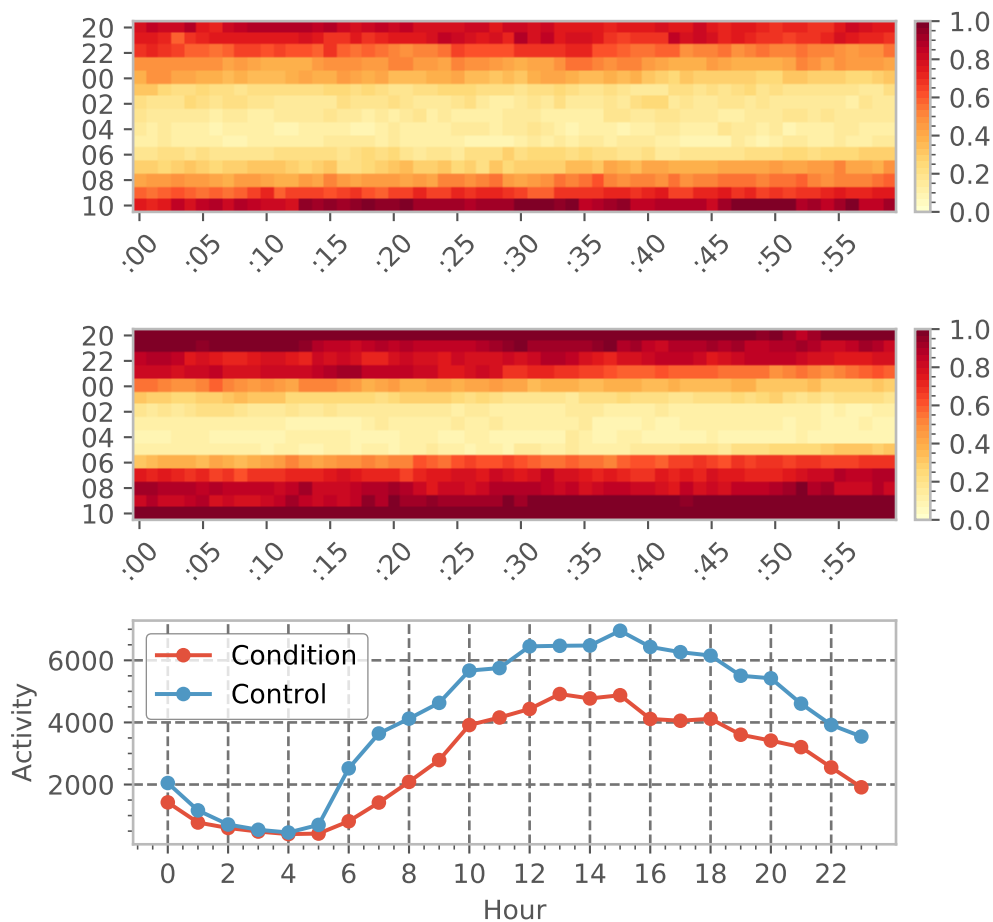


Figure 3.6: Activity intensity average for all patients. The top two figures show minutely activity levels during the night from 8pm to 10am, where the irregularities in sleeping patterns among depressed patients (top figure) and reduced activity levels is more distinct. The middle figure shows the same for the control group, whereas the last indicates the total activity count on hourly basis.

The activity count for the control group is notably higher in the evenings and in the mornings, and the overall sleeping patterns are more consistent. Further on, to analyse how this pattern is on a weekly level, we generate a heatmap to observe the average intensity over a week with a hourly distribution, as shown in **Figure 3.7**.

The overall indication shows that activity levels on weekly basis is also clearly less for the depressed group, where in terms of sleeping pattern, the depressed patients also

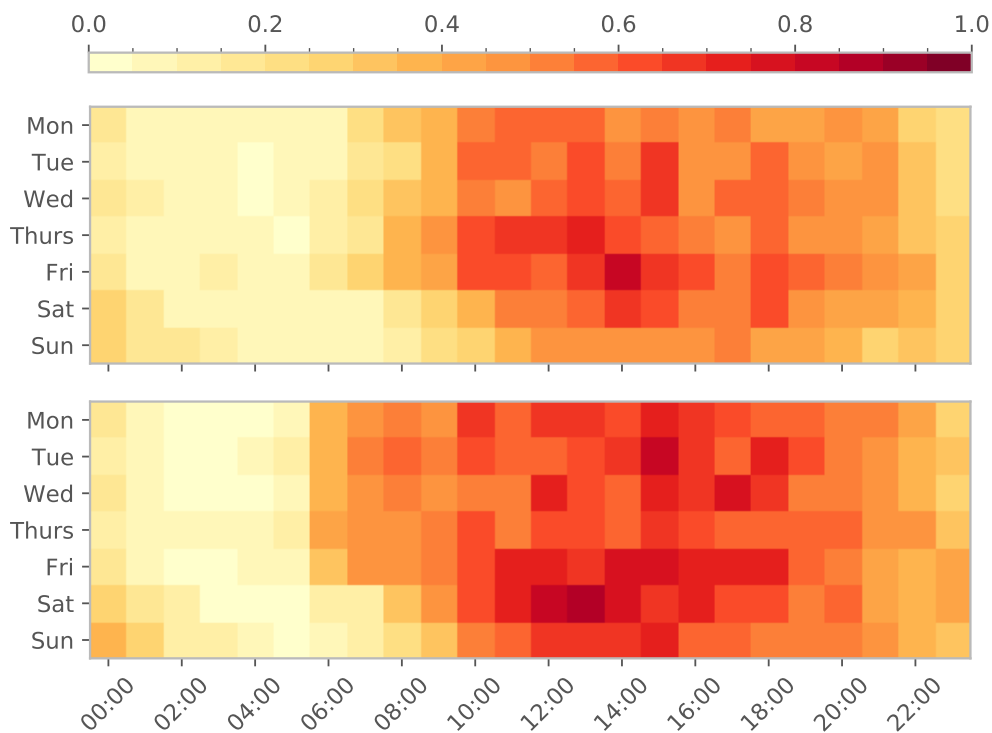


Figure 3.7: Heatmap of the total hourly activity over a week, averaged over all patients. The top figure shows an activity heatmap for depressed patients, whereas the bottom figures illustrates the same for control patients.

sleeps in longer. Not only does this analysis show that depressed patients are less active on a general level, but sleeping patterns are also less consistent with increased irregularities. To understand the activity cycles and the temporal dependencies, a correlation analysis is performed.

The autocorrelation function (ACF) is used to analyse the temporal correlation in activity levels. Based on the earlier analysis the initial hypothesis is that depressed patients have less consistent activity patterns, further evident in **Figure 3.8**, which shows the correlation coefficients for multiple resamples of the dataset. The figure shows multiple correlation plots with different resampling frequencies, but explains the same pattern. In this case, multiple resamples are used to illustrate how the average level activity correlation decreases over longer timespans,

With an hourly-, 8- and 12-hour based resampling the overall correlation shows a cyclic trend, indicating that the correlation in daily activities is more positive, hence a higher correlation for specific time periods. Further on, the observations suggests that over longer timespans, the activity levels for the depressed group is less consistent, similar to earlier observations. Hypothetically, this is may be related to irregularities in day-to-day activities compared to the control group. The inconsistency is also more notable when observing the correlation over daily activity levels. Nevertheless, The analysis suggests that analysing timespans of approximately daily or weekly activity levels should capture the temporal dependencies well.

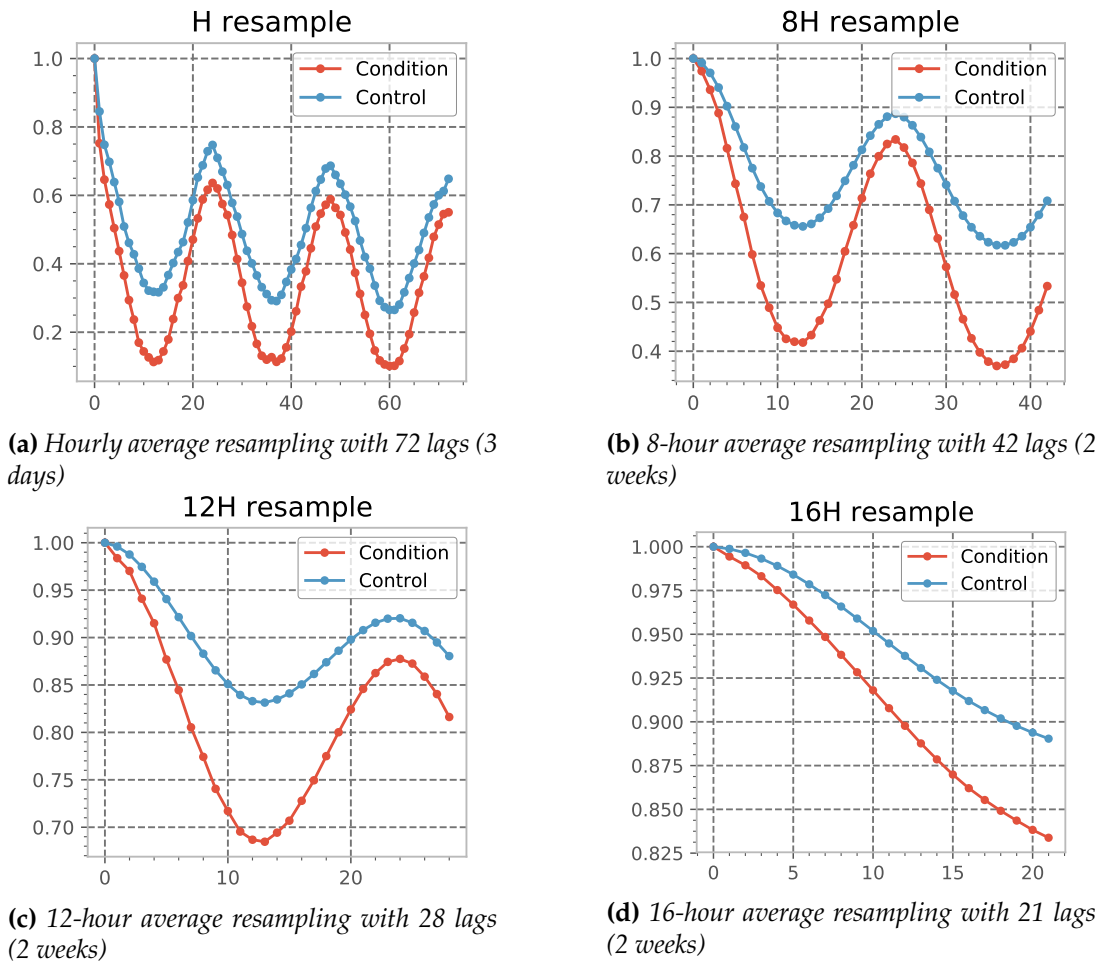


Figure 3.8: Correlation coefficients from ACF that shows temporal correlations with various lags for different resampling averages of motor activity levels. Each lag represents the average activity level within the specific time period, hence the correlations show the average pattern across all patients for both groups.

3.3.3 Summary of use case

In this section, we first presented a general level medical background for U_1 , the depression detection use case. We defined the term depression, various forms and the application of machine learning as an approach for depression detection.

Further on, we presented the depression dataset [91] and provided an analysis using visualisations and correlation analysis. In this analysis, we showed how activity levels differ between the depressed and non-depressed groups on average. Overall, we saw that depressed patients sleep in longer, have less consistent activity levels and show increased irregularities in sleeping patterns during the night. The irregularity pattern and inconsistency in activity levels were further evident in the correlation analysis. The analysis showed that the correlation coefficient decreased faster for the depressed group on a daily level, which indicate that the activity levels vary over time.

3.4 U_2 : Energy prediction

3.4.1 Energy background

Renewable energy

In recent years, focus on using renewable energy sources has increased. This is mostly a response to the goal of achieving a more sustainable future in regards to energy consumption and production. A recent report from the International Renewable Energy Agency (IRENA) [73] includes data from approximately 200 countries, both from official and unofficial sources. Statistics in the report suggests that the overall trend globally is increasing in terms of generation and production from renewable sources. The investment in renewable sources are also increasing, and the trend is not only observable in industrialised countries, but is similar in developing countries.

Electric vehicle ownership

Global EV Outlook [31] is an annual publication that discusses the global state of electric vehicles (EVs). It is published by the International Energy Agency (IEA) and includes research on a multitude of countries that are members of IEA, mostly industrialised countries. According to the report from 2018 [31, 32], there is an increasing number of EV ownership in the world and the growth rate since 2015 is over 50%. The interest in general is also increasing, especially in countries like Norway because of supportive policies and incentives provided by the government discussed by IEA in their 2018 report.

The power grid

As the report mention, because of this innovation of electrification, there is an increased demand for energy. In general, one important implication is the discussion on how the power grid responds to the increased demand of EV charging, such as how to avoid grid overload and reduce cost for grid upgrades. These are primary problems, because of how the power grid is designed today.

The traditional power grid is based on a one-directional model from generation on a power plant to distribution among consumers. The generation of the power is decided in advance with prior knowledge on demand profiles, which usually varies between residential, commercial and industrial users. In general, as IEA describes, there are peak hours during a day where the demand is at its highest. This usually affects the electricity prices and how the power is generated, with minor possibility for flexibility in the power grid.

An important motivation is how we can overcome these problems by distributing the energy demand over time. Not only will this affect the demand which influences the energy prices positively but the distribution of demand reduces the power load during peak hours.

Energy optimisation and machine learning

As described earlier, the increase in energy demand requires a flexible power grid where load distribution is optimised. Distribution companies, such as Hafslund in Norway experiment on developing grid systems to determine how to reduce cost and provide a sustainable infrastructure for increased energy demand [36].

However, the research and experimental studies done on EVs specifically are far less as it is a recently trending field. Sundstrom and Binding [86, 87] discusses in some of their papers on how to optimise charging processes with traditional optimisation methods, such as linear and quadratic optimisation. With an increased popularity around EVs and some research in energy and EV charging optimisation, there is still relatively limited research on how machine learning models are applicable for this optimisation problem. The field has recently emerged in recent years, similar to the popularity around EVs and applied research using machine learning approaches are thus limited.

3.4.2 Dataset

Overview

The EV dataset is a multivariate time-series with hourly frequencies and contains observations on percentage battery levels, temperature, odometer⁴, vehicle brand and charging state. The dataset is originally intended for research and development on energy optimisation techniques and energy demand prediction.

Car ID	Observations	Days	Period
1	4395	199	28.10.18 - 15.05.19
3	4239	187	09.11.18 - 15.05.19
4	4170	181	15.11.18 - 15.05.19
6	3893	169	27.11.18 - 15.05.19
7	3856	167	29.11.18 - 15.05.19
5	3503	150	16.12.18 - 15.05.19
11	1740	96	08.02.19 - 14.05.19
10	1955	84	08.02.19 - 02.05.19
12	755	67	09.03.19 - 15.05.19
Total	9	28506	1300 28.10.18 - 15.05.19

Table 3.3: Overview of the EV dataset. The table shows statistics on number of vehicles, collection period in days and number of data points both before and after cleaning. We see there is an increase data points after cleaning, indicating that certain hourly observations were initially missing.

The data is collected from individual participants through a system developed as part of another project. During time of writing, data is still being collected with additional

⁴An odometer is the device/instrument that is used for measuring travel distance on cars.

cars registered gradually. Overall, the dataset contains observations on 9 different cars in total, from two different brands. Namely, Tesla and BMW. Data collection for the first car dates back to October 2018, whereas the most recent car includes observations from March 2019. Due to data still being collected, this thesis uses the latest extraction of the dataset, briefly summarised in **Table 3.3**.

Data analysis

To understand the underlying data distribution and form hypotheses prior to initial model configurations, a visual overview and analysis of the dataset is performed. First, a prior assumption is related to usage patterns of EVs, which can be explained through driven distances, similar to regular cars. Second, specifically for EVs, charging patterns and battery levels may also be an equally good indicator of future usage behaviour, like daily commutes and activities in general. To better understand this, the relative percentage battery levels are analysed. Further on, driving distances are also analysed through odometer data.

Figure 3.9 shows the distribution of hourly battery levels across a week with a heatmap. Although there are variations in the relative battery levels, we observe a notable pattern. Evidently, many cars are charged during work hours on weekdays and not surprisingly in the evenings and over the night, despite some EVs are not. For instance, car 5, 6 and 7 charge more often during work hours and 1, 10 and 11 show a more regular pattern of charging during the evenings and over night. This observation corresponds well with usage patterns in general. Evening activities may contribute to decrease in capacity, whereas charging also may be economically beneficial at work, if charged at the workplace.

Nevertheless, the indication is more frequent cyclic patterns throughout the day although we observe a more general level pattern on a weekly basis. We derive this from the decrease in battery throughout the week, whereas the capacity increases again at the end of the week for most of the cars. However, this is not the case for all EVs, such as car 4 and 12. To further understand this temporal pattern, we perform a correlation analysis using the autocorrelation function (ACF), similar to earlier in **Section 3.3.2** for our depression detection experiments. Arguably, we want to understand the dependencies in time and correlations on how often charging is present which is implied by the increase in battery levels. We perform this analysis on 4 different resampling averages, illustrated in **Figure 3.10**.

Although the figure shows multiple correlation plots with different resampling frequencies, the subfigures in essence explain the same pattern. With 24 lags with an hourly frequency there are no regular patterns in battery levels. However, there is a notable temporal structure with longer lags, of approximately 1 week. With 4-hour average there is an increasing correlation and we observe a cyclic pattern which we discussed earlier, possibly explaining the charging cycles. Similarly, we do this for the 8-hour resampling and daily average, hence, we argue that sequence windows representing weekly battery levels captures the temporal dependencies well.

Despite this analysis explaining correlation on average battery levels, we are aware how certain driving and charging patterns affect the outcome of this analysis. From

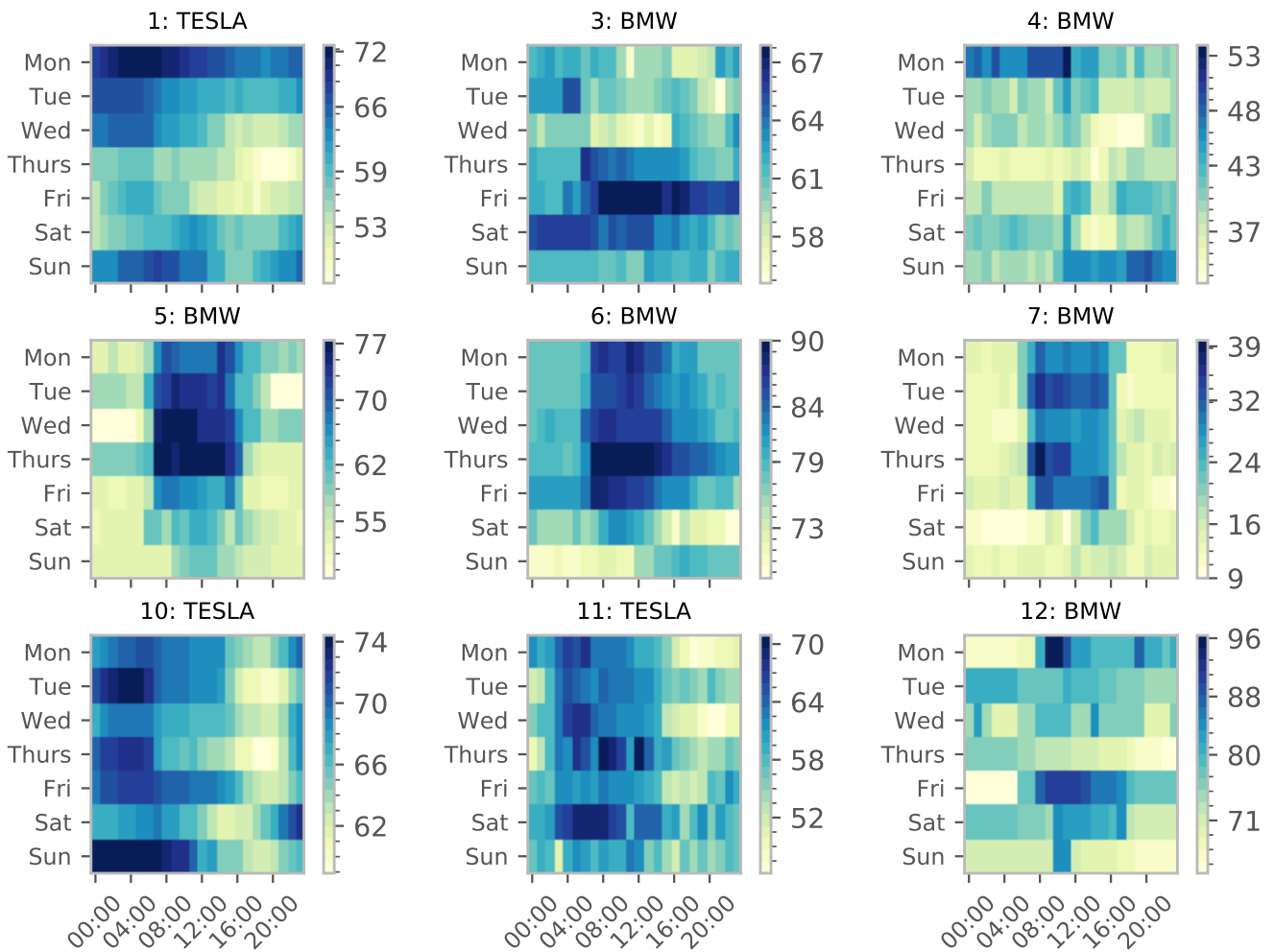


Figure 3.9: Weekly battery level distribution for each car with an hourly frequency. The percentage battery level relative to each car shows variations in overall usage, and some cars are charged far less than others, possibly being hybrid EVs. Car 12 is an exception due to the timespan of observations, where the collected number of days is less, hence the smaller intensity in the heatmap.

the heatmap we presented earlier, we observe certain outliers. Cars like number 4 and 7 have battery capacities no more than 40%-50%, and we argue this may be related to types of vehicles, such as hybrid EVs.

Notably, we do know that hybrid EVs are present in the dataset, although we are unaware of which ones. Nevertheless, one implication of this is a different charging and driving pattern, in which we observe the prior in our heatmaps, especially for car 4 and 7 where we see irregularities. There are higher variations, and higher battery levels are less frequent throughout the week. Hence, it is of importance to account for this when predicting the energy capacity. As we can see, percentage battery levels would not be a sufficient predictor, and is dependent on both type of vehicle but as we argue, the vehicle brand is also of importance as it gives an indication of absolute capacity used.

For instance, the electric range for a Tesla model S varies from 400 to 600 kilometers

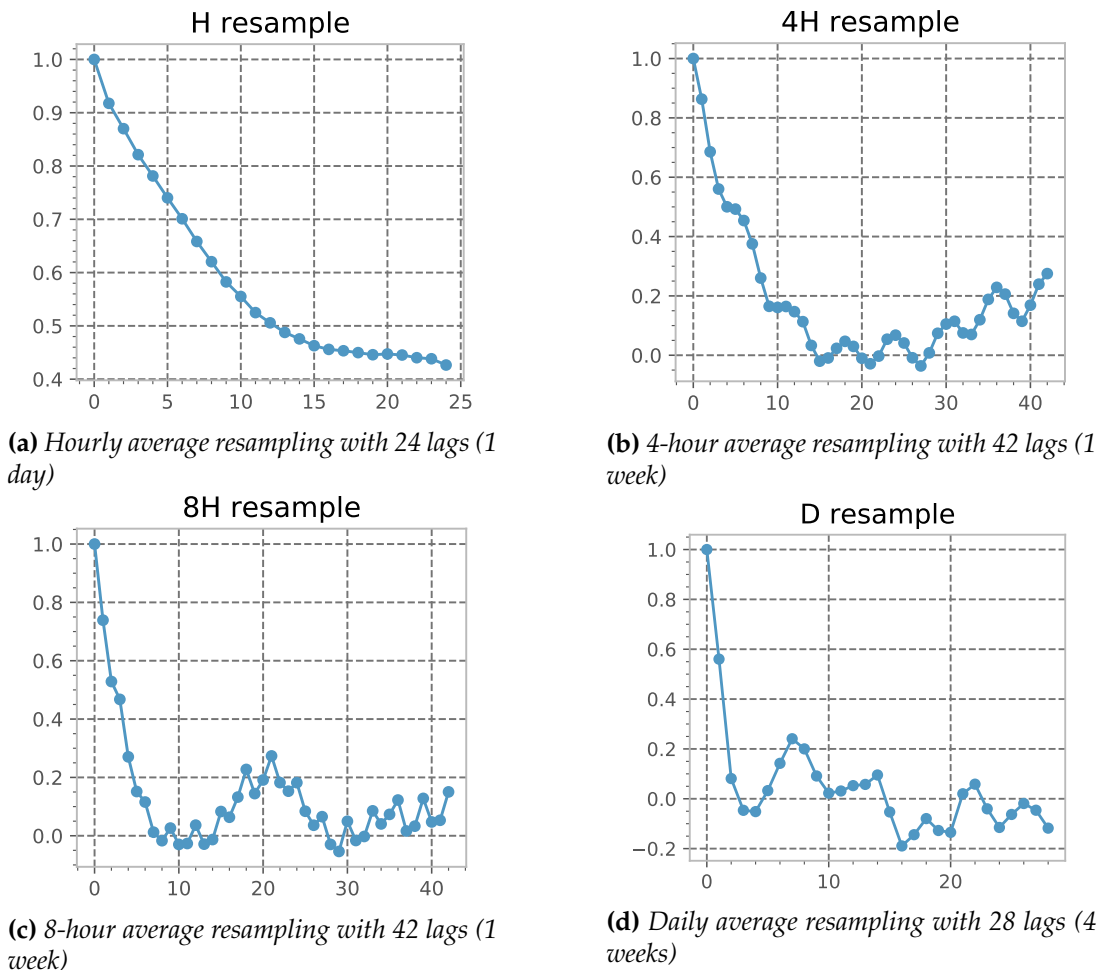


Figure 3.10: Correlation coefficients from ACF that shows temporal correlations with various lags for different resampling averages of battery levels. Each lag represents the average battery level within the specific time period, hence the correlations show the average pattern across all EVs.

depending on total battery capacity, whereas the distance capacity for a BMW i3 ranges from 100 to 200 kilometers, both being the only types in the dataset. The EV dataset only contains the brand name and as such, the absolute capacity range for each car is unknown. However, with the absence of this information, we aim to better explain the mentioned variations related to hybrid EVs and vehicle brands by looking at the driven distances and battery level distribution. Performing such an analysis allows us to better understand which cars are most representative in our dataset. Firstly, we observe the distribution of battery levels using histograms, as illustrated in **Figure 3.11**.

Notably, an even distribution across all bins indicates both higher charging frequency and usage frequency. We argue this because more observations on the lower end indicates low battery levels, hence frequent usage. Similarly for the higher end with higher battery levels, indicating frequent charging. A distribution which is right-skewed however, indicates higher battery levels, and therefore higher charging frequency with possible less usage. However, another explanation to a right-skewed

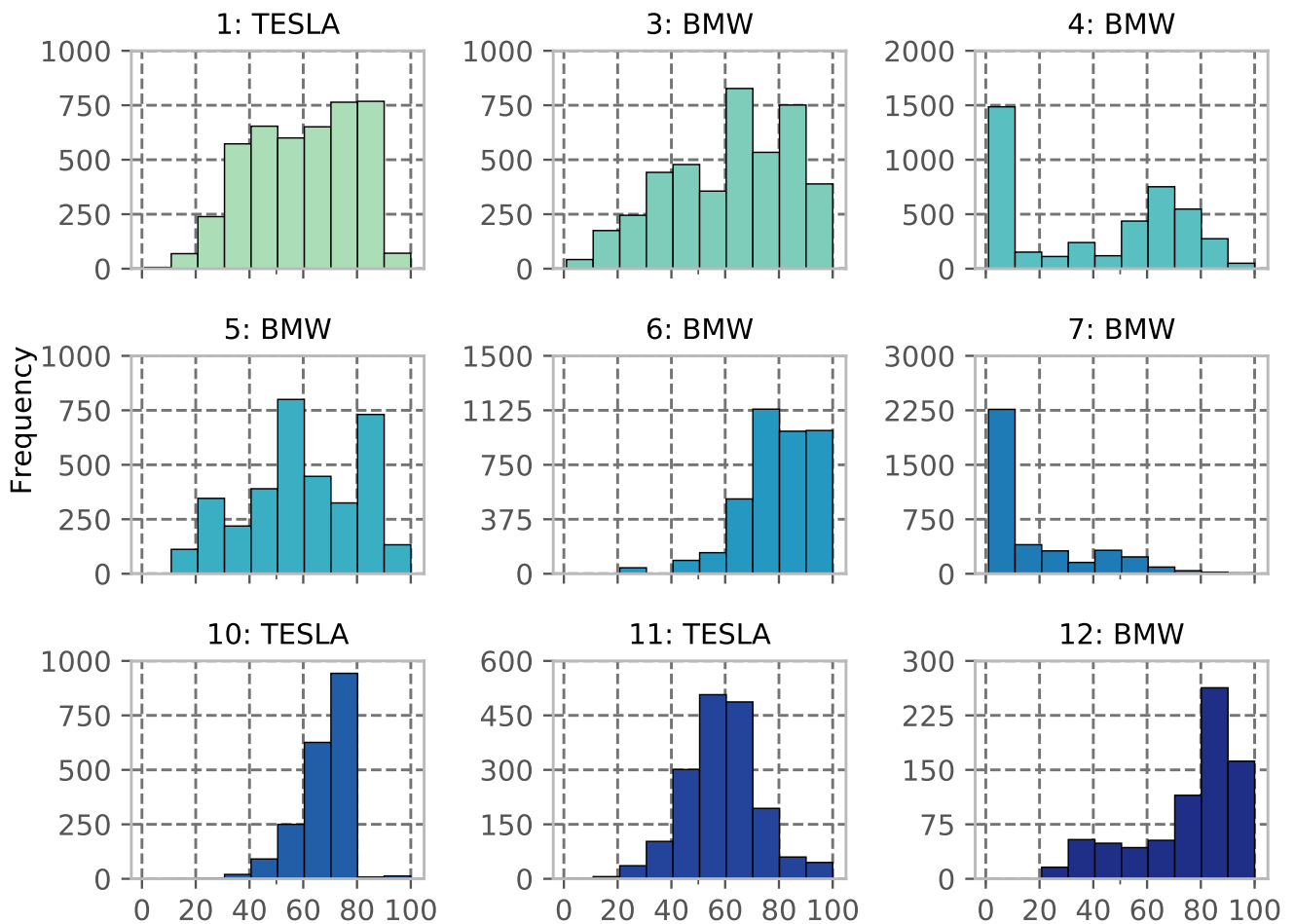


Figure 3.11: Histograms for all vehicles in the EV dataset, annotated with vehicle ID and brand. The X-axis denotes the hourly percentage battery levels, intervalled into 10 bins, whereas the Y-axis denotes the frequency of battery levels within each bin, which is equivalent to a 10 % interval.

distribution is also frequent usage, in which the usage is often shorter distances driven. Hence, access to chargers is more probable. A left-skewed distribution would explain the opposite, with less charging but high usage, for instance in the case of hybrid EVs where charging can be less intense.

Ideally, an even distribution would be the most representative for cars that are used and charged often. For instance, vehicle 1 has evenly distributed battery levels, indicating both frequent charging and usage. This assertion may hold for vehicle 3 and 5, whereas this is not the case for vehicle 6. The distribution of this car is right-skewed, and as we argued it may be an indication of less usage. On the contrary, it may indicate higher usage in terms of shorter distances driven, but possibility of charging is higher between each shorter trip. Comparably, vehicle 4 and 7 have left-skewed distributions. For most observations the range of battery level is no more than 10%, although vehicle 4 has some observations with higher battery levels, potentially explaining longer trips. As we discussed above, these vehicles may be hybrids.

Nevertheless, recall that we initially described hybrid vehicles as potential outliers in

our analysis, due to less frequent charging patterns. We also see this from the left-skewed distributions in the histograms in **Figure 3.11**. Additionally, we also argued why EV brands is of importance for energy prediction, because it gives an indication of absolute capacity. Our analysis suggests that there is a particular variability in general and percentage battery levels are only relative measures for battery capacity. We emphasise however, that additional features should be included for the modelling problem, to better capture the underlying distribution. This may include features such as actual used capacity for each observation. Despite this information being unavailable in the EV dataset, there is a possibility of including it based on the available car brands, although this is more of an estimate based on model capacity. We further explain choice of input features in **Section 4.2.1** about experiment overview, but aim to provide an additional analysis on the driven distances to better explain the usage patterns, which gives an indication on day-to-day driving behaviours in general.

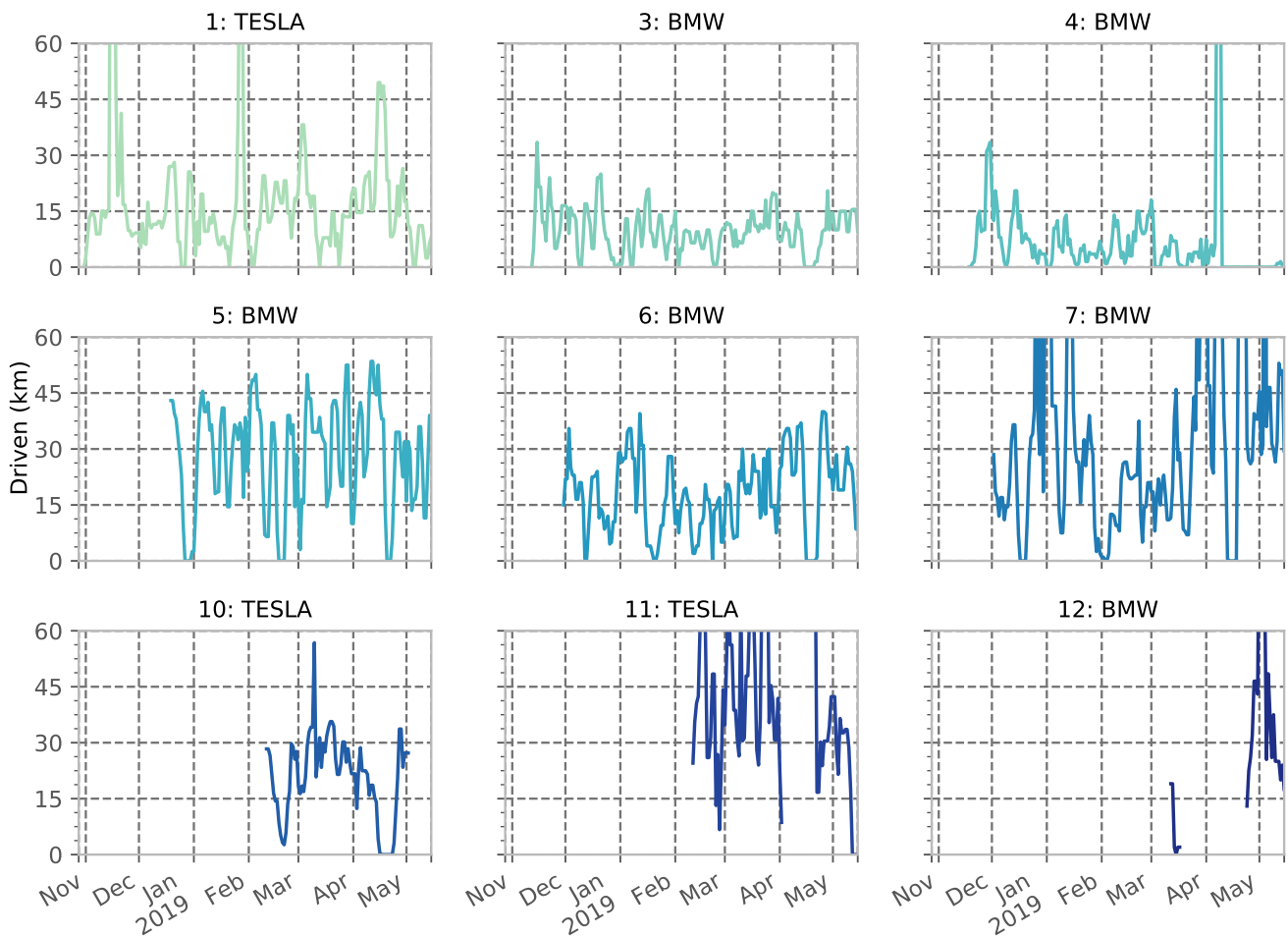


Figure 3.12: 4-day rolling median driving distances for all EVs in the EV dataset. The Y axis denotes the driven distance in kilometers, whereas the X-axis denotes the timespan. We constrain the Y-axis to 60 kilometers and the figure shows that only some cars drive more than this. For car 11 and 12, there are also missing observations in the period April-May.

Figure 3.12 shows the median level driving distance on a 4-day rolling basis. Arguably, median level provides a better indication on usage, compared to the

average which may be subject to outliers like longer trips. The overall suggestion is that driving distances have higher variations across all cars. The median driving distance on a 4-day basis is 15 to 30 kilometers for most cars although some vehicles are used more. The cyclic patterns correspond day-to-day usage, whereas the spikes are possible longer trips. For car 1, 3 and 4 the general driven distance is around 15 to 20 kilometers. Vehicle 5, 6 and 7 however, shows higher variations possibly explained by day-to-day usage like commuting being longer.

3.4.3 Summary of use case

To summarise, we first presented a general level background of the energy domain in the context of EVs. We discussed renewable energy has been trending in recent years, how the ownership of EVs has increased drastically, and the effect they have on the traditional power grid. Lastly, we presented various approaches regarding energy optimisation and how the application of machine learning is relatively new in the context of energy prediction and optimisation.

Furthermore, we presented the EV dataset with an overview and data analysis. The analysis showed how battery levels between EVs vary throughout the day. Some people charge more during work hours, although in most cases, EVs are charged during the night or early in the morning. Overall, the charging cycles were shorter on a daily level and longer on weekends, which we also observed in the correlation analysis. Hence, EV users charge daily, but in shorter periods, whereas the vehicles are charged longer by the end of the week.

Moreover, we performed a simple analysis of the battery level distribution and driving distances, through histograms and line charts. The distribution in battery levels suggested high variations between EVs. Some were used more frequently than others, but the distribution of battery levels was left-skewed, which we argued could be hybrid vehicles.

3.5 U_3 : Football readiness classification

In this section, the last use case is presented and discussed. First, we look at a general level background of performance optimisation in the context of the football domain. Furthermore, the dataset is discussed with corresponding data analysis, before we summarise the resulted insights.

3.5.1 Football performance background

Performance optimisation and football

Performance in sports is not an unknown concept, and it is a general understanding that professional athletes train to optimise their performance. Top athletes use a lot of time and effort to be better, considering it is part of their daily life. Factors

contributing to optimal performance often include proper restitution, sufficient nutrition and intensive workouts [72].

However, in recent years, with the advancement in technology, sensors and wearables, another aspect is concerned with the application of data-driven and analytical methods to boost performance [53, 92]. Most wearables and sensors today can detect physiological signals like heart rate and respiration frequency. With such information quantified, the potential application areas within performance enhancement are diverse.

For instance, the ability to predict peak readiness in training [94] based on self-reports is beneficial both in terms of optimal performance, but also tactical decisions. Similarly, there exist tools for game analysis to optimise game-play as well, such as the *Forzify LiveCam*, which is built on the real-time system *Bagadus* [85].

Emerging technologies and innovation hubs

Data-driven approaches and adaption of technology in football has only emerged in recent years. However, because it serves a competitive advantage, football teams invest more in innovations that advance player and team performance. For instance, FC Barcelona, one of the largest football clubs in the world, has an innovation hub [7]. The facility focus on advancing research within various fields in the organisation, like performance optimisation, health and restitution and technological applications [92]. Similarly, Liverpool FC is also one of many clubs investing in technological innovations in football [81].

Nevertheless, such organisations possess quantitative data on performance from the best football players in the world. However, it is not easy to acquire such information in general without having an existing framework. Hence, the dataset used for this particular use case presents multiple opportunities in the domain of readiness prediction.

3.5.2 Dataset

Overview

The dataset used for this use case is part of an existing performance monitoring system called PMSys [69]. It is a mobile application built on the real-time system *Bagadus* [85] and was developed by Simula Research Laboratory, University of Tromsø and ForzaSys. As of today, various teams are using or have used the PMSys-system, including multiple Norwegian football teams and the Norwegian national team as well.

Data collection happens through individual questionnaires submitted through a mobile application by players. In total, the current dump of the dataset includes quantitative data from 36 different players, where the series contains observations from late 2016 and early 2017 to September 2019. Each player records ratings on

mood, stress, soreness, fatigue, sleep quality, sleep duration and perceived exertion (PE).

Description	Count (raw)
Players	27 (33)
Fatigue	5911 (6006)
Mood	5911 (6006)
Perceived Exertion	5911 (6006)
Readiness	5911 (6006)
SleepDuration	5911 (6006)
SleepQuality	5911 (6006)
Soreness	5911 (6006)
Stress	5911 (6006)

Table 3.4: Overview of the PMSys dataset [69]. The table shows total observations for all variables in the dataset and the total number of players, both before and after cleaning. In total, the dataset includes data on 33 players, although 6 of them are excluded, conditioned on the number of observations being less than 31 days.

The variables are numerical ratings, mostly on the same scale from one to five. In each category, one is the lowest rating, whereas five is the highest. For instance, a mood rating of one indicates a bad mood and a score of 5 on sleep quality implies sufficient restitution [94]. However, PE is an exception from this scale, ranging from one to ten and explains the perceived form of the player.

While PMSys is a system already in production, the potential application areas of the collected data are valuable in various ways. Not only is the information useful for performance improvement, but additional insights can be used to better plan training sessions or make tactical decisions. One such use case, for instance, as discussed by Wiik et al. [94], is predicting peak readiness-to-train, which showcase the possibility of understanding when players can perform optimally.

Data analysis - understanding correlations

This section presents the initial findings and analysis of the PMSys dataset. Primarily, the analysis is limited to extracting insights from player readiness and stress levels. Mostly, because the classification tasks focus on predicting readiness and stress levels. Because the dataset includes multiple variables, the initial analysis is concerned with understanding the overall correlation between them. Correlation coefficients can be visualised through a correlation matrix, as illustrated in **Figure 3.13**.

The correlations are generated based on the average scores across all unique players in the dataset and thus explains the variable relationships on average. Lighter colours represent a weaker or negative correlation between a variable pair, while a darker colour indicates the opposite.

Further on, when considering readiness as the target variable, a positive correlation can be seen when compared to the variables *Soreness* and *Fatigue*. In a way, these observations are reasonable, considering how energy levels and muscle soreness

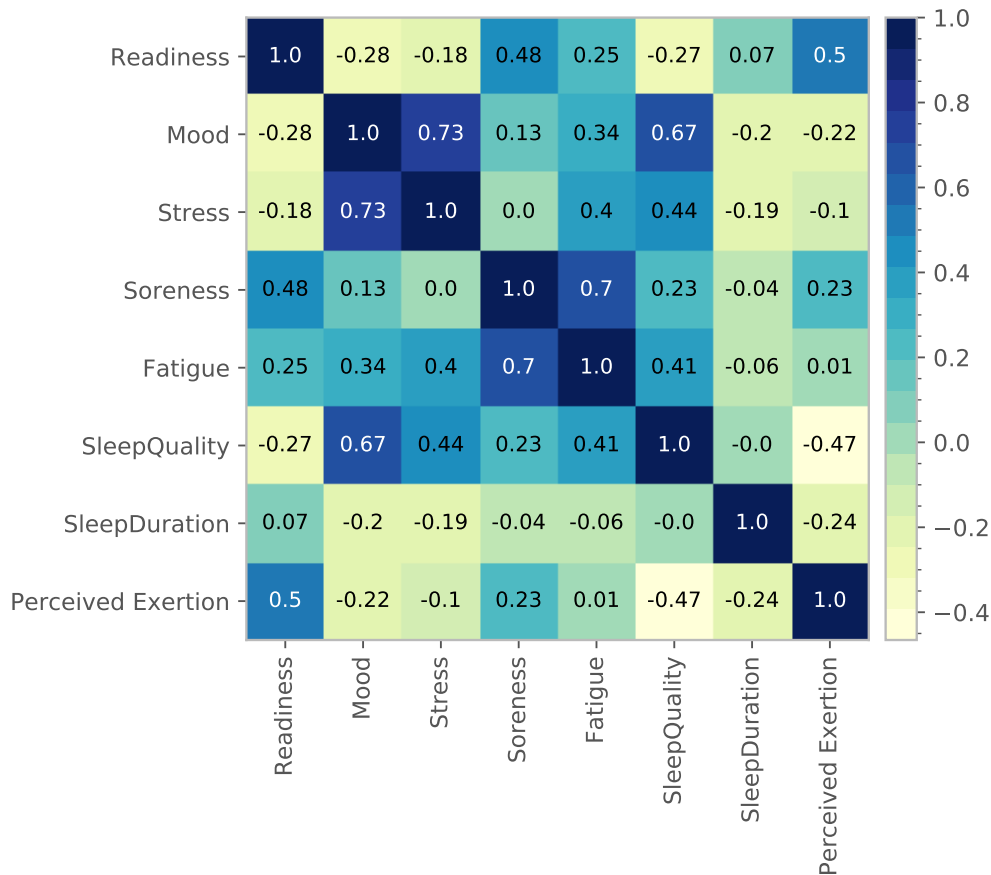


Figure 3.13: Correlation matrix of variables in the PMSys dataset. The matrix shows the correlations between each variable pair. Because the correlation between a given variable V_i and V_j is symmetrical, the coefficients below and above the diagonal are the same.

potentially affects physical effort. On the other hand, when considering stress as the target variable, there is a positive correlation between mood, fatigue and sleep quality. A higher score on the stress level implies that a player is very relaxed, and the mood, fatigue and sleep quality seems to correlate positively with this. Hence, a good mood, low fatigue and better sleep quality reduce the overall stress level among players.

Although these variables positively correlate with the target variables, there are other indicators which are negatively correlated as well. For instance, the relationship between readiness and mood is -0.28 , which indicate that the readiness of players is affected negatively by their mood levels. Similarly, the sleep quality of players affects their readiness negatively. Lastly, when looking at the stress levels of the players, the sleep duration looks to be the only factor contributing to increased stress. However, in general, for both the stress and readiness scores, the negative coefficients are closer to zero, which may indicate weaker correlations. Nevertheless, they may still be contributing input features in the classification task itself.

Data analysis - understanding temporal correlations

Although the analysis above discusses the correlations between variables directly, it does not consider the temporal relationship. Hence, the correlation C_{ij} between the i -th and j -th variable explains the relationship between both variables at the same time step t . As outlined in **Section 3.1.1** in the methodology-chapter, the autocorrelation function (ACF) is applied to understand temporal correlations in variables.

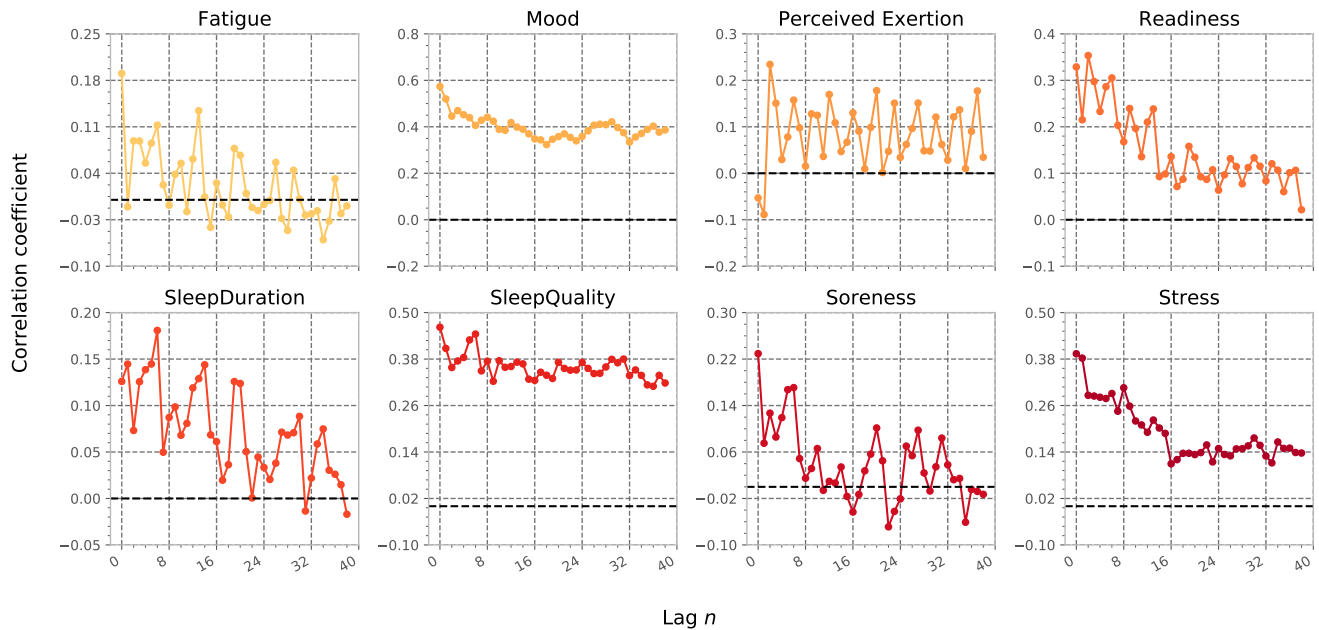


Figure 3.14: Correlation coefficients for each variable in the PMSys dataset up to 40 days lag. The black dashed line highlights zero. Overall, decreasing coefficients indicate that scores for a given variable over longer periods are less relevant, as seen for stress, soreness and readiness correlations. Slower decreasing coefficients imply that the scores reported longer back in time are closely correlated with recent reports, as can be seen from the sleep quality and mood coefficients.

Figure 3.14 shows the correlation coefficients on an average level, for each variable up to a 40-days lag, thus, from $t - 40$ up to timestep t . While it may seem that the overall coefficients are relatively low, the temporal dependencies for each variable emit different patterns. For instance, it looks like there are higher variations over time for variables like fatigue, perceived exertion, sleep duration, readiness and soreness. In general, the coefficients are decreasing over time, implying how factors like fatigue, sleep duration and soreness from multiple days back have little or no correlation to the levels reported in more recent days.

Moreover, mood, sleep quality and stress are almost constant factors over longer timespans. Compared to the abovementioned variables, the coefficients are also higher, which may indicate that these factors are explanatory factors for optimal performance over time. For instance, a constant higher correlation in mood and sleep quality indicates on an average level, that most players have a good mood and feel like they sleep better over an extended period. Another aspect, however, is how correlation in sleep duration decreases over time, where sleep quality is more or less

constant. While it can explain better restitution among players, it may as well explain how sleep duration has a smaller impact on quality. Further on, one implication is how sleep quality is potentially explained better by other factors than only sleep duration.

Note that the correlation analysis performed in this and the previous section is only concerned with understanding the temporal patterns. It is emphasised, however, that understanding the significance in the correlations and pairwise correlations are of importance. However, due to time and resources constraints, we consider this outside the scope of this analysis.

Data analysis - underlying distributions

The last analysis is concerned with understanding the underlying distribution of reported readiness scores. In terms of a classification perspective, it is arguably important because categorical variables like readiness may contribute to imbalanced scores.

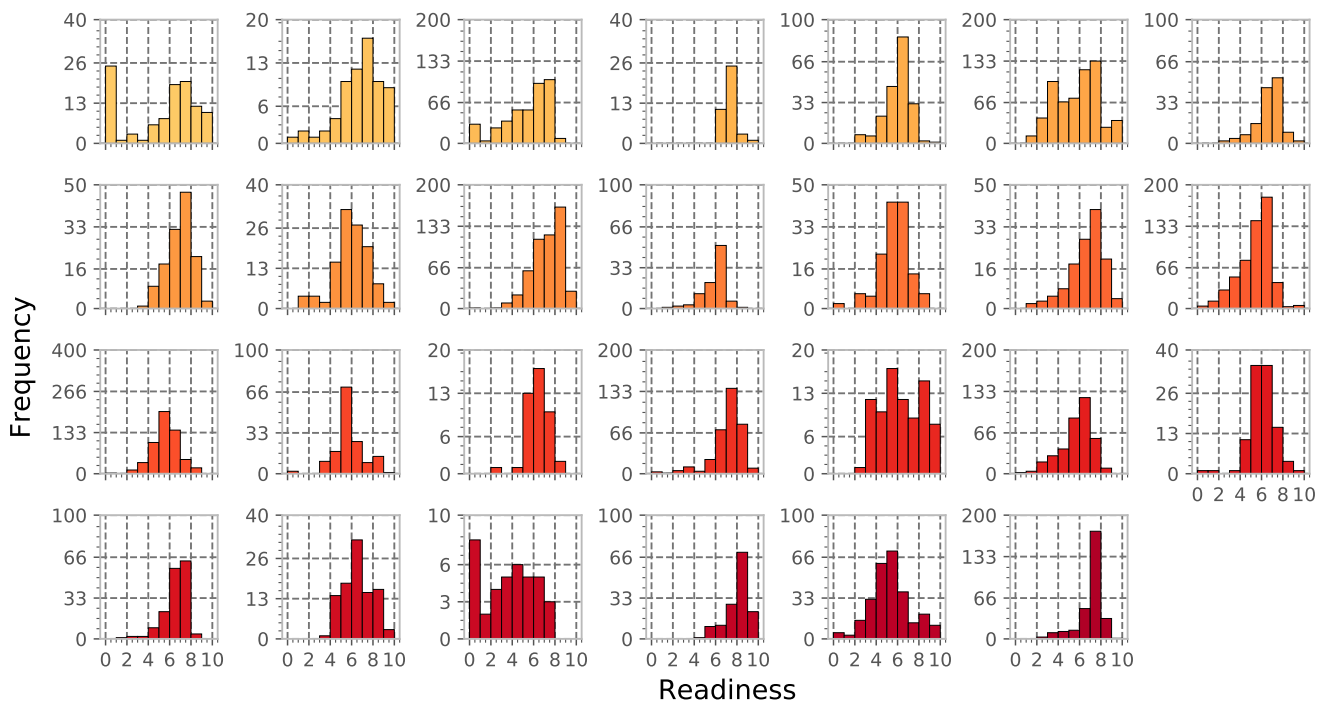


Figure 3.15: Distribution of readiness scores for all players in the PMSys dataset. The X-axis denotes the readiness scores intervalled into ten bins, one for each score. The Y-axis denotes the frequency for each bin. Overall, the figure shows how most reported readiness scores are evenly distributed around five, where most variations range from three to eight.

As illustrated in **Figure 3.15**, which shows the distribution of reported readiness scores for all players, there is one notable observation. Most scores range from three to eight with an even distribution around the centre score of five. The implication is an average reported readiness score for each player being around five, which is of importance when formulating the classification task. There is an underlying class

imbalance in terms of readiness scores.

For instance, consider a binary classification task where readiness is thresholded to a score of seven. According to the distribution shown above, many of the classes reside in the *not ready* class. However, from a performance and coach decision perspective, such a classification model may not be a useful contribution. Most specifically, because a score of seven can still be a reasonable score in terms of readiness, despite classified as *not ready*. Hence, this also shows why developing a model that outputs a quantitative measure [94] is beneficial, compared to a qualitative measure and crisp output.

Nevertheless, one interesting aspect is the possibility to predict multiple qualitative outcomes based on historical reports. For instance, one formulation of the classification task is applying a multilabel-classification model⁵ which outputs a set of classes. In this case, a multilabel-classification model may be a significant contribution, as it outputs a set of classes for a given player. However, it is still of importance to determine the threshold of the reported scores in developing such a model.

3.5.3 Summary of use case

In this section, we first discussed the background in terms of football performance optimisation. We shortly described performance optimisation in football and how recent advances in technology, sensors, wearables and data-driven analysis have been applied to enhance performance. Further on, we presented how emerging technologies and innovation hubs are used in two larger football clubs in Europe to advance research within the field of performance optimisation.

Moreover, we presented the PMSys dataset [27] and corresponding data analysis. First, we performed a simple correlation analysis between each variable on an average level. We observed how soreness, fatigue and perceived exertion positively correlated with readiness scores. On the other hand, mood and sleep quality had a negative correlation. Further on, to we analysed the relationship in time with the ACF, to understand the temporal structure for each variable. We showed how mood, sleep quality and stress scores were relatively constant over time with positive coefficients. We argued these variables could be a contributing factor for optimal performance over extended periods because of this consistency. At last, we gave an overview of the distribution of readiness scores for each player in the dataset. It was evident that the mass of readiness scores was centred around five, and ranged from three to eight in most cases.

⁵A multilabel-classification model is a classification model where the output distribution includes multiple classes and not one class. One example is predicting the presence of a feature from a set of categorical variables like stress, readiness, mood and fatigue, e.g. understanding if a player is stressed and in a bad mood but still ready.

3.6 Model development

In related work, we discussed how various applications of deeper CNNs and RNNs had successful applications across various domains in recent years. Although deeper networks can explain their efficiency, we purposely keep the depth of our networks small and shallow. While doing so, it presents a better comparison basis for both architectures. Because deeper models introduce additional complexity, a comparative study of both architecture would be more difficult as well. It is, however, important to emphasise that different architectures may be more efficient than others given the context of their applications. For instance, if shorter sequences are used, vanilla RNN or simpler models may potentially perform better compared to LSTM.

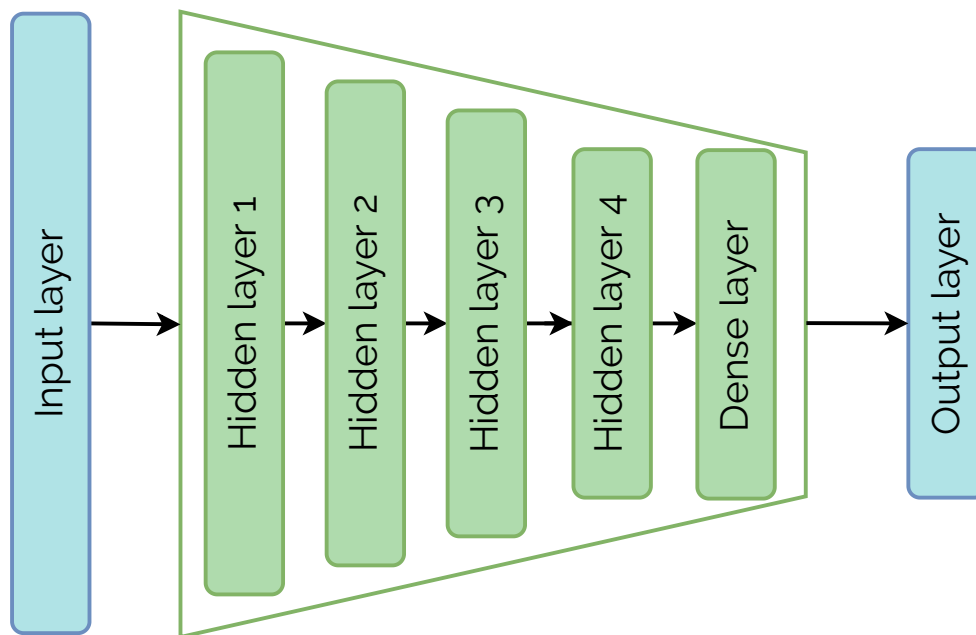


Figure 3.16: A general overview over the network architecture. The input layer represents the input units and the hidden layers are stacked on top of each other with the same architecture type, like LSTM or CNN units. Lastly, the dense layer represents the optimisation layer with all interconnected nodes and the output layer represents the predictions. In total, the base architecture have 5 hidden layers.

Figure 3.16 illustrates the general architecture of both models. In total, there are five hidden layers, motivated by other comparative studies using shallow networks [5, 93], where most use less than ten hidden layers. The first four hidden layers are of the same type with convolutional layers or LSTM-units. We do not use pooling layers for CNN to better understand the effects of the convolution operation. Further on, we apply dropout after each hidden layer which is considered good practice to improve generalisation and prevent overfitting. The last hidden layer is a fully connected (dense) layer. For U_1 and U_2 , the softmax activation function is applied to generate a class distribution. For U_3 , the multilabel classification problem/football use case, we use the sigmoid activation function to determine the class based on probability thresholds.

Moreover, the number of units in each layer is also kept relatively low to reduce model

complexity. The default number of units in each hidden layer are 150, 75, 50, 25 and 25, respectively. We first chose these settings through trial and error, but throughout this thesis, the topic has changed drastically. The current setting that we use is based on the initial experiences and various ranges used in similar comparative studies of CNN and LSTM [5, 98].

Further on, the number of nodes in subsequent layers reduces throughout the networks. They also remain fixed for all experiments, similar to the number of layers. This design was determined based on two motivations. First, various neural network architectures like CNN-models in the image classification domain apply pooling to downsample and reduce spatial dimensions. The first few layers detect high-level features with more hidden nodes, and the last layers are constrained to detect more significant low-level features. Second, as shown in **Figure 3.5**, a fixed number of hidden units for both LSTM and CNN, makes them more comparable in terms of the model architecture and the number of trainable weights can explain their complexity. Although total weights are smaller for our CNN, the general network architecture itself is the same as the LSTM, with the same number of hidden nodes and layers.

Model architecture	Trainable parameters
LSTM	193 052
CNN	102 002

Table 3.5: Overview of total number of trainable parameters for the specified architecture. LSTM-model has 91 050 more optimisable weights than the CNN-model. Note that this is in context of the default architecture of five hidden layers. Each of the hidden layers of the same unit type has 150, 75, 50 and 25 units, respectively, with the last dense layer having 25 units.

3.7 Model training and hyperparameters

Our model training and experiment evaluation is separated into two parts. First, based on the data analysis and recent research, we explored initial hyperparameters, to understand how the models learn the data distribution without the application of early stopping [9]. We tried various configurations, somewhat through trial and error and based on best practices, particularly for most network-related hyperparameters. The approach was more systematic for data-related hyperparameters like batch size, window size and resampling frequency, which relied more on the data analysis. Second, based on observations during experiments with these initial results, we further chose a subset of hyperparameters for model optimisation with the application of early stopping.

3.7.1 Model training

Listing 3.1 shows how we train the model with cross-validation and the initial part of the experiment flow. We first format the data and generate configuration-related logs and directories and then save the configurations. We keep track of the current

```

:
19 def main(config, data):
20     X, y = utils.format_data(config, data)
21
22     print('|_Creating_log_configurations_... ')
23     config.create_log_configurations()
24
25     print('|_Saving_log_configurations_... ')
26     config.comments = f'{config.comments},_{str(X.shape)}'
27     config.save_config()
28
29     folds = config.data['folds']
30     current_best_acc, current_best_mcc = 0, -2
31     best_fold = None
32
33     print(f'|_Training_model_with_{folds}_folds_... ')
34     folder = KFold(n_splits=folds)
35     exp_evaluation = experiment_history.EvaluationHistory(config=config)
36     exp_training = experiment_history.TrainingHistory(log_dir=config.log_dir)

```

:

Listing 3.1: *Subset of run.py that shows the experiment initialisation process*

best accuracy and MCC to determine the best model but save all the models during training. Further on, we have implemented two classes that keep track of training- and validation history. Before cross-validation, we create an instance of both and update the history during training and evaluation. We then start the cross-validation process. First, we split the dataset and build and compile a model instance for each iteration through our implemented model-interface. We then train the compiled model and evaluate it.

Further on, **Listing 3.2** shows the flow of the cross-validation, model training and evaluation. For the initial experiments, the models run for a given number of epochs. For the optimised experiments, on the other hand, we apply early stopping, which is a technique to prevent overfitting [9]. We do this through the **EarlyStopping** class in Keras and monitor the validation loss and validation accuracy. The validation loss acceptance threshold is ten iterations, and the validation accuracy threshold is set to the max number of epochs.

Upon termination, we save the trained model for the given cross-validation. To keep track of the best performing model, we check the current best metric and determine whether or not to replace the existing best. We save all models for all validations and keep a separate instance for the best performing one. At last, the training- and evaluation history are saved.

```

    ⋮
37
38 for k, (train_index, test_index) in enumerate(folder.split(X)):
39     print(f'|_X:_{X.shape}\n|_y:_{y.shape} ')
40     x_train, x_test = X[train_index], X[test_index]
41     y_train, y_test = y[train_index], y[test_index]
42
43     # Build model
44     model_handler = model.Model(config=config)
45     md = model_handler.build_model(x_train.shape)
46     start, used = 0, 0
47
48     try:
49         # Train model
50         start = time.time()
51         history = model_handler.train_model(x_train, y_train)
52         used = time.time() - start
53         print(f'|_K_{k+1}_|_Used_{used:.2f}_seconds ')
54     except KeyboardInterrupt as e:
55         break
56
57     # Evaluate model
58     print('|_Evaluating_model_on_test_set... ')
59     predictions = md.predict(x_test)
60     is_football = config.usecase == config.FOOTBALL
61     evaluations = None
62     if is_football:
63         evaluations = exp_evaluation.custom_evaluate(predictions, y_test, k +
64             1, data['columns'])
65     else:
66         evaluations = exp_evaluation.evaluate(predictions, y_test, k + 1)
67     exp_training.update_history(history.history, used)
68
69     # Save model
70     print('|_Saving_model... ')
    md.save(f'{config.log_dir}/models/k{k+1}.h5')
    ⋮
```

Listing 3.2: *Subset of run.py that shows the model training process, from cross-validation splitting and model compilation to model evaluation.*

3.7.2 Hyperparameter selection

U_1 : Depression detection

Table 3.6 gives an overview of all the hyperparameters we tried in the initial and optimised experiments for this use case. Recall that our analysis showed how temporal patterns were most distinct when observing a time span of approximately one week. Hence, one of the most important settings initially was to explore a combination of sequence window length and resampling frequency which was approximately equivalent to this period. We tried various configurations that represented a one- or two-week time span. Furthermore, the batch sizes were configured through trial and error and were adjusted based on the observations in

Model Exp	CNN						LSTM					
	I_1	I_2	R_1	R_2	R_3	R_4	I_1	I_2	I_3	R_1	R_2	R_3
Activation	relu	relu	relu	relu	relu	relu	relu	relu	relu	relu	relu	relu
Batch	10	20	15	15	15	15	8	10	15	5	5	5
Dilation Rate	1	4	4	4	4	4	-	-	-	-	-	-
Dropout	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Kernel Size	10	6	6	6	6	6	-	-	-	-	-	-
Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.005	0.005
Momentum	0.1	0.1	0.1	0.3	0.1	0.3	0.7	0.7	0.1	0.7	0.5	0.4
Nesterov	False	False	False	False	True	True	False	False	False	False	False	False
Optimiser	sgd	sgd	sgd	sgd	sgd	sgd	sgd	sgd	sgd	sgd	sgd	sgd
Resample	12H	10H	10H	10H	10H	10H	10H	10H	12H	8H	8H	8H
Window	14	35	30	30	30	30	28	28	28	30	30	30

Table 3.6: Overview of tried hyperparameters for U_1 , the depression use case, where the initial experiments are denoted with I .

the experiments.

In terms of network-related hyperparameters, many of the configurations were kept constant. For instance, the optimiser, activation and learning rate were not explored extensively. In the initial experiments, the convergence and learning process was desirable. Hence, we decided to keep SGD with the learning rate of 0.001, with ReLU as activation. However, throughout the experiments, hyperparameters like momentum were explored to optimise this convergence. For the CNN in particular, we tried to tune the kernel size and dilation rate to see the effect on how they captured the temporal dependencies.

U_2 : Energy prediction

Model Exp	CNN			LSTM		
	I_1	R_1	R_2	I_1	I_2	R_1
Activation	relu	relu	relu	relu	relu	relu
Batch	1	1	1	5	8	15
Dilation Rate	3	3	3	-	-	-
Dropout	0.3	0.2	0.2	0.3	0.4	0.4
Kernel Size	8	5	5	-	-	-
Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001
Momentum	0.3	0.3	0.3	0.3	0.3	0.1
Nesterov	False	False	False	False	True	False
Optimiser	sgd	sgd	sgd	nadam	sgd	nadam
Resample	D	D	D	D	D	D
Window	30	30	30	30	30	21

Table 3.7: Overview of tried hyperparameters for U_2 , the energy use case. Initial experiments are denoted with I .

For the data-related hyperparameters in the energy use case, the series is fixed to a daily frequency. The dataset is relatively small, and many hourly observations are already missing. As seen in **Table 3.7**, we decided to focus on the classification of

usage with a daily frequency, because this would be more representative for capacity usage. Based on the data analysis from earlier, it was evident that charging patterns occurred daily in smaller cycles and for longer periods on a weekly level. To capture these cycles effectively, we tried using a larger time span with a sequence window length of 30 days. For the batch size, we tried on-line training for the CNN with a batch size of 1. **Appendix B.4** shows some results with various batch sizes. Many of these often resulted in either bad convergence or no improvement at all when we tried only tuning the batch size. For the LSTM, on the other hand, this hyperparameter did not affect the outcome in a similar manner, where we saw that larger batch sizes worked more efficient.

For network-level hyperparameters, we tried various configurations, mostly to prevent overfitting and optimise both performance and convergence. During the initial experiments, both models were sensitive in the sense that various configurations gave results with large contrasts. Some are shown in **Appendix B.4** and **Appendix B.3** showcasing efforts where convergence would be really slow or not converge at all. To overcome this problem, we tried exploring different optimisers and Nesterov momentum. For the CNN in particular, the initial results were more desirable, and various configurations showed that the model would be easier to explore in terms of hyperparameter optimisation. Hence, the settings from the initial results were kept, and we tried two experiments with the same configurations.

U_3 : Football readiness classification

Model Exp	CNN			LSTM	
	I_1	R_1	R_2	R_1	R_2
Activation	relu	relu	relu	relu	tanh
Batch	15	15	15	15	10
Dilation Rate	3	3	3	-	-
Dropout	0.3	0.3	0.3	0.3	0.3
Kernel Size	4	5	5	-	-
Learning Rate	0.005	0.005	0.005	0.001	0.001
Momentum	0.4	0.4	0.2	0.4	0.4
Nesterov	False	False	False	False	False
Optimiser	sgd	sgd	sgd	rmsprop	rmsprop
Resample	D	D	D	D	D
Window	30	30	30	30	40

Table 3.8: Overview of tried hyperparameters for U_3 , the football use case. Initial experiments are denoted with I .

Table 3.8 shows the hyperparameters that were tried for this use case. Most importantly, while we tried some initial experiments with LSTM, the results were very similar to those presented, and thus, were chosen to be excluded. The only data-related hyperparameters explored were sequence window length and batch sizes, similar to U_2 , because of the series having a daily frequency. As suggested by the data analysis, the temporal correlation had higher variations over time for particular

variables like fatigue, soreness and readiness scores. To understand these variations, we let the sequence window be approximately one month, similar to what is applied by Wiik et al. [94] in their study. Moreover, the initial values of 10 and 15 on batch size seemed to give the best results and good model convergence, so we chose to use these particular values.

Moreover, also shown in **Appendix B.5**, we tried multiple optimisers, activation functions and different values of momentum rate and learning rate. For the CNN, the convergence was already good, but momentum was tried to reduce the overall fluctuations. For the LSTM, many configurations for the initial experiments did not show much improvement, and as described, the results were very similar. The LSTM was relatively consistent during the training process, so the chosen hyperparameter ranges were those who worked optimally in most of the cases.

3.8 Experiment evaluation

We evaluate the experiments with 10-fold cross-validation. The resulting confusion matrices of the predicted outcomes are used to derive prediction metrics, described in the following sections. We evaluate each result with weighted average metrics and determine overall performance by averaging across all folds.

3.8.1 Evaluation methods

K-fold cross-validation

Models are evaluated with K-fold cross-validation [45] as a baseline to assess the results for independent datasets. K-fold is a desirable method of validating models because data partitioning is usually binary. One fixed partition is held out as a test set for evaluation whereas the other fixed partition is used for training, the training set. In datasets with class imbalances, a fixed partitioning does not give a good indication of predictive performance, as the proportion of classes may have an uneven distributed across partitions.

K-fold cross-validation [45] is an evaluation method that partitions the dataset into K equally sized folds. Each fold is used as a test set during the cross-validation process, completely held out during training. All other folds are used as training data, in which a subset of the training data is used for validation during training. Overall performance is determined by averaging across all folds. This gives a better evaluation as the variations between binary partitions could fluctuate more, depending on the partition of the datasets.

We use 10-fold cross-validation, where each fold is a set of formatted sequences with a specified window length of size W . Training and test set is partitioned to 90/10, where 10% of the data is used as a test set. Moreover, a subset of the training data is used as a validation set during training, and the overall partition scheme is 80/10/10 for train/validation/test.

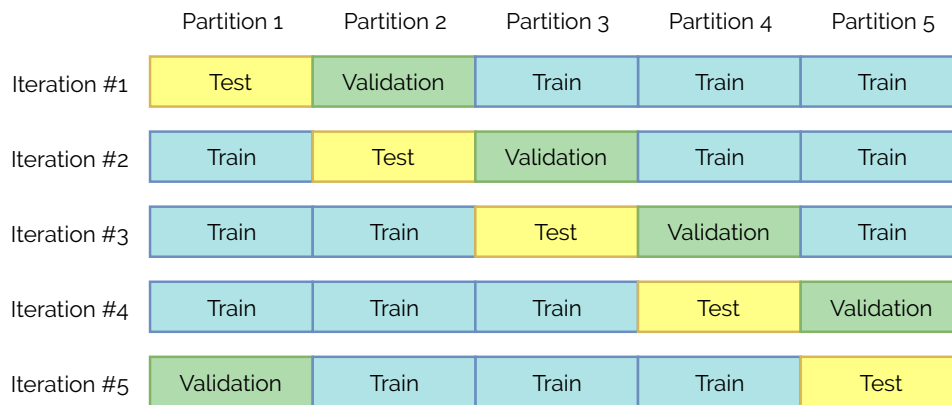


Figure 3.17: Example on dataset partitioning with 5-fold cross-validation. Each fold is used as a test set and a subset of the training data (rest of the folds) is used as a validation set, to validate the model performance during training.

Confusion matrix

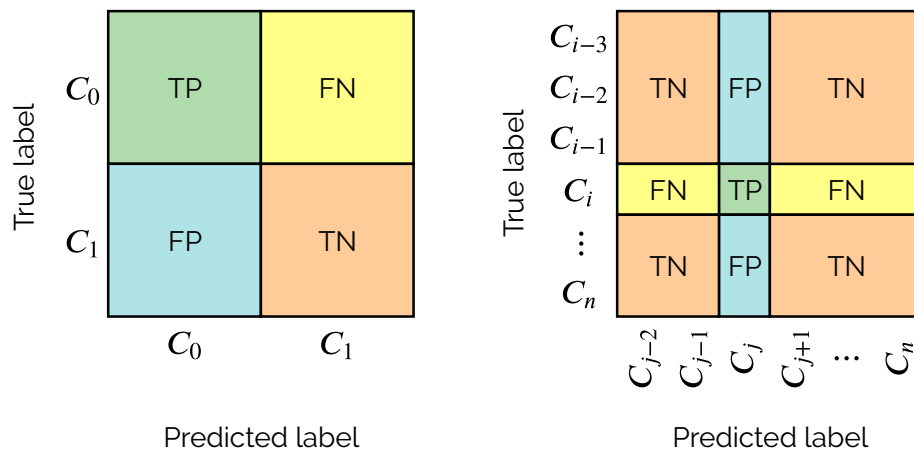


Figure 3.18: The left figure shows a confusion matrix for binary classification whereas the right figure shows the same for multiclass classification with n classes. In a binary classification problem there are two classes, a positive and a negative class in which a correct classification is considered as a true positive or true negative, respectively. In multiclass problems, the true positive is understood in relation to the predicted class C_j and true class C_i .

The above section described how K-fold cross-validation is used to evaluate the models. Before looking at metrics used for evaluations, we discuss how these are derived and interpreted.

The generated output predictions of a test set evaluation can visualised with a confusion matrix. A confusion matrix [45] is a matrix comparing predicted classes against the true classes/labels. More formally, an entry C_{ij} in the confusion matrix denotes the j -th predicted class for the i -th true class. Ideally, a well-performing model has a higher ratio of predicted instances where $i = j$.

As shown in **Figure 3.18**, confusion matrices for each test set evaluation can be used to derive multiple descriptive measures like true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). The measures are used for computing different evaluation metrics, as we further describe in the next section.

3.8.2 Evaluation metrics

We use accuracy (ACC), specificity (SPEC), precision (PREC), sensitivity/recall (REC), F1-score (F1) and Matthews correlation coefficient (MCC) to evaluate results. In the following sections, an overview of the metrics is presented with the case of binary classification as a motivation.

Accuracy

Accuracy (ACC) is the percentage of correctly classified samples in relation to all samples and is the most common metric for measuring model performance. However, it does not give an accurate measure of the performance and can be a faulty measure. For instance, training on imbalanced datasets where positive samples outweigh negative samples or vice versa, it affects the accuracy and does not give an overview of overall performance.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.1)$$

Equation 3.1: *The accuracy metric.*

Consider the example of a dataset with ten positive samples and 90 negative samples in the context of classifying tumours. If all cases of non-tumours (negative class) were correctly classified, whereas all cases of tumours (positive class) were misclassified the accuracy is 90%. However, this defeats the intention of correctly classifying tumours, and on the contrary, if all tumours were correctly classified and all non-tumours misclassified, the accuracy would be 10%. Overall, this metric does not tell us how well the model performs in classification tasks with imbalanced classes.

Precision

Precision is a metric concerned with measuring the relevance of predictions. It measures the rate of positively predicted classes among all classes predicted as positive, hence its alternate name, positive predictive value. For instance, although there may be certain situations where all positive classes are correctly classified. High misclassifications for positive classes (false positives) may impact the relevance.

$$PREC = \frac{TP}{TP + FP} \quad (3.2)$$

Equation 3.2: *The precision metric.*

Consider the case where 30 out of 60 cases of non-tumours are predicted as tumours (30 false positives), but all actual cases of tumours were correctly classified as such. Intuitively, the rate of relevant predictions would be lower as there are many classified cases of irrelevant non-tumours. We want to find the tumours, and in this case, the precision rate would be $\frac{10}{10+30} = 0.25$.

Specificity

Specificity is also known as the true negative rate. It is a measure of the negative class and tells us the rate of correctly predicted negative classes, among all negative classes. Similar to the accuracy, it should be used in combination with other metrics. For instance, if the number of negative classes outweighs the positive classes, the specificity rate is higher.

$$SPEC = \frac{TN}{TN + FP} \quad (3.3)$$

Equation 3.3: *The specificity metric.*

Consider the same example we presented earlier with tumour classification. If the total number of predicted non-tumour cases were 60, there are 30 cases of misclassifications (the false positives), where non-tumours were classified as tumours. This would result in a specificity rate of $\frac{60}{60+30} = 0.66$.

Sensitivity

Sensitivity is also referred to as recall. Compared to the specificity rate, this metric measure the true positive rate, which is the correctly predicted positive classes, among all positive classes. The sensitivity rate can be seen jointly with the specificity rate we defined above. If the specificity rate is high (true negative rate), the sensitivity rate (true positive rate) will be lower and vice versa. This is due to the increase in misclassifications of either classes.

$$REC = \frac{TP}{TP + FN} \quad (3.4)$$

Equation 3.4: *The sensitivity metric.*

In context with the example, we defined for specificity, say 7 of 10 tumour-cases was classified correctly and the rest were misclassified as non-tumours. The sensitivity rate is then $\frac{7}{7+3} = 0.7$. Notably, in classification tasks with imbalanced datasets, and also in general, it is desirable to achieve a higher sensitivity rate and specificity rate jointly. This implies that the prediction rate for negative classes *and* positive classes is good.

F1-score

Although many of the metrics we presented above depend on the classes and are sensitive to class imbalance, the F1-score is a joint metric, that considers class imbalance. It provides a balanced measure by looking at the precision and sensitivity.

$$F_1 = 2 \cdot \frac{PREC \cdot REC}{PREC + REC} \quad (3.5)$$

Equation 3.5: *The F1 metric.*

Matthews correlation coefficient

Matthews correlation coefficient (MCC) is a scalar value between -1 and 1, indicating the strength of prediction. A positive value implies positive predictions and agreement with the observations (where 1 is a perfect agreement with the observations). Similarly, negative values imply negative predictions and disagreement with the observations. An MCC of 0 implies that the prediction is random, and there is no agreement or disagreement with the observations. Generally, MCC is considered as a balanced measure suited well for imbalanced classes, similar to the F1-metric.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (3.6)$$

Equation 3.6: *The MCC metric.*

3.9 Summary of methodology

Conclusively, the aim is to answer the question *How do CNNs compare to LSTMs for time-series classification?*. This research question is motivated by how various applications of CNNs for time-series and sequential data have shown to be promising in recent years, compared to RNNs. Comparing both architectures requires the implementation of two different models.

In this thesis, we compare CNN with LSTM, where the Python programming language is used for implementation. Further on, Keras is used for model development with the GPU-release of Tensorflow as the backend. Lastly, for exploratory data analysis, descriptive statistics and figure generation, the functionalities of Pandas are applied.

Moreover, the developed CNN and LSTM are tested on three different use cases and datasets, where two of them are multivariate time-series and the last is a univariate time-series. The models are used for classification of depressed patients, energy prediction for Electric Vehicles (EVs) and football player performance classification. Furthermore, the complexity of the networks is relatively low. They are sparse in terms of depth. The number of hidden units within each layer is fixed throughout the

experiments as well. Each model is applied separately for each use case, evaluated accordingly and optimised afterwards.

Chapter 4

Experiments

The previous chapter described the methodology where we presented the experiment framework and overall flow, use cases and data analysis. Further on, we discussed model development and training and provided a discussion on hyperparameters before discussion evaluation methods. This chapter discusses the experiments and results in detail. For each use case, we present an experiment overview followed by results for each model and a comparison of both. At last, we present a summary before providing a general-level discussion across all use cases at the end of this chapter and summarise the findings.

4.1 U_1 : Depression detection

4.1.1 Experiment overview

Overview of narrative

The analysis from earlier suggests observing approximately daily activity levels to learn overall temporal patterns. Additionally, it is evident that the intensity in activity is less consistent among the depressed patients, hence higher variations on an average level. Based on this analysis, initial hypotheses were formed about the data distribution on how input sequences are formatted. Although multiple results are presented, both for LSTM and CNN, a general level overview is discussed for all experiments.

With basis in the general scheme for the experiment design illustrated in **Figure 1.3** earlier, a brief narrative is shown for this use case in **Figure 4.1**. There are six experiments for each model, separated into initial/exploratory experiments and optimised experiments. Each result is discussed, although an in-detail comparison of LSTM and CNN is discussed in **Section 4.1.4**. In total, there are 3 and 2 initial experiments for LSTM and CNN, thus 3 and 4 optimised results for each, respectively.

Each initial experiment is concerned with trying multiple combinations of hyperparameters through trial and error to see how each model performs with different configurations. We first start with data-related hyperparameters like the resampling fre-

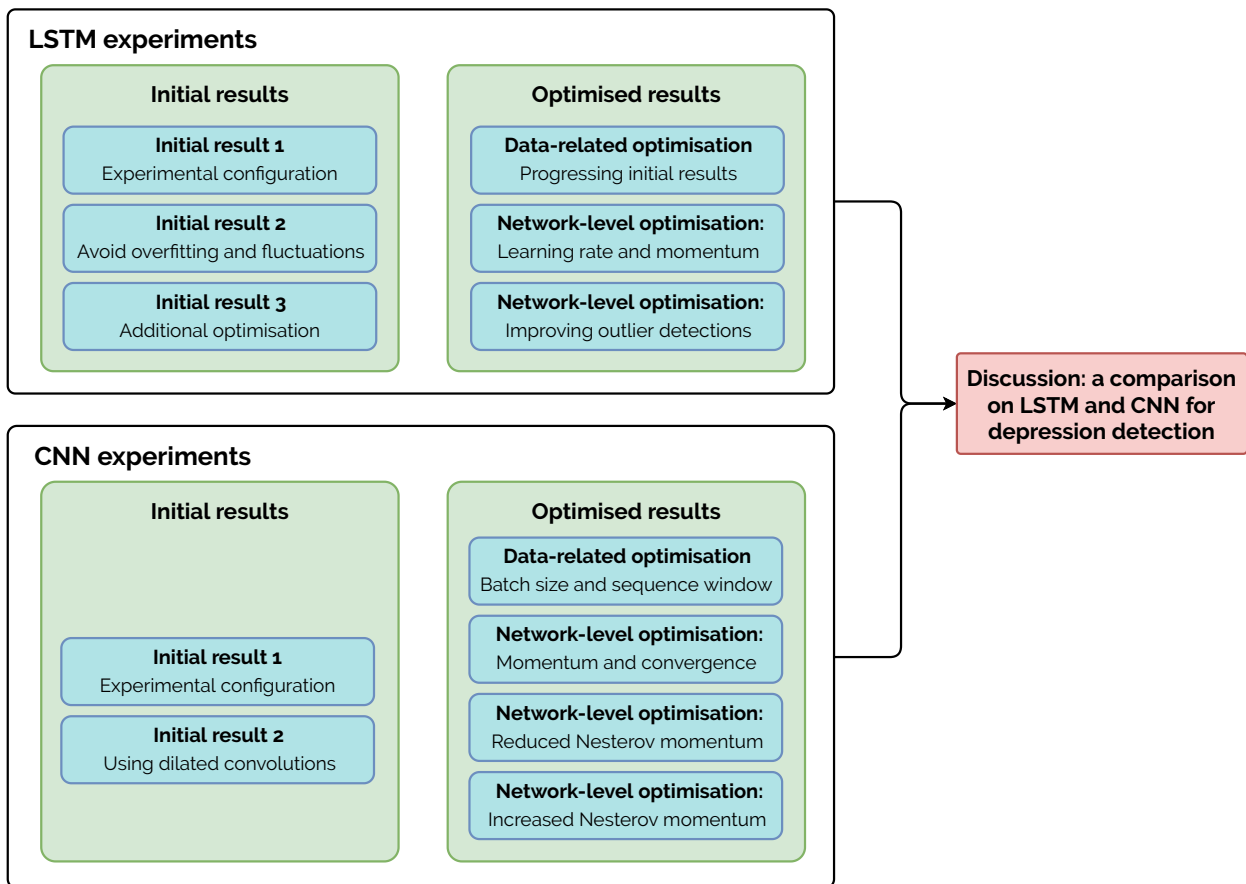


Figure 4.1: General level experiment overview for depression detection experiments. There are 3 initial experiments and 3 optimised experiments for LSTM, whereas 2 initial and 4 optimised experiments are performed for CNN.

quency, sequence window and batch size, which are set based on the analysis earlier. Further on, the models are optimised by tuning certain network-level hyperparameters. This includes momentum and learning rate, adjusted with basis in default values set in Keras. As a naming convention throughout the narrative, we use R_n to denote the n -th given optimised result, because the initial results are mostly concerned with exploration.

Overview of experiment preparation

In this section we discuss use case specific choices made throughout the experiments. How data is preprocessed, formatted and how input features are selected for the models will be discussed.

Firstly, because the series contains activity counts with a minutely frequency, we justified how resampling of the time-series helps in understanding the temporal pattern earlier. Additionally, the data files are shuffled prior to reading and resampling. This introduces a certain stochastic element for the models, although

it may give a better representation of samples/batches that are used.

Secondly, deviational measures are used as additional input features. The total mean activity level and standard deviation is used, similar to what is discussed by Garcia-Ceja et al. [29] in their paper. However, they also include the percentage of events with no activity and motivate for how discriminative features are applicable for better classification. Either of these features are included, because many of the initial experiments were performed using only mean and standard deviation as additional features.

Lastly, being aware of how deviational measures varies over different timespans we suggest rolling windows could be applicable, like mean or standard deviation over a given timespan compared to the entire series. Although rolling windows are not used, it should be considered as an alternate approach regarding feature selection, in addition to inclusion of discriminative features. Arguably, this may be more representative compared to overall activity deviations and further experiments can be carried out. Conclusively, in terms of feature scaling and normalisation, being a univariate time-series we choose not to scale the input features as the numerical scale is the same. However, this is also a possibility regarding preprocessing, as it is a common practice when training neural network models [20].

4.1.2 Results for LSTM

I_1 : first initial result

Type	Hyperparameter	Value
Data	Batch Size	8
	Epoch	300
	Resampling Average	10H
	Sequence Window	28
Network	Dropout	0.3
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.7
	Nesterov Momentum	False
	Optimiser	sgd

Table 4.1: Most important hyperparameter configurations for the initial results.

Through all initial and exploratory experiments, the models are trained for longer periods purposely to understand how the model fit to the data distribution. The training iteration (epochs) is set to 300, although for this particular use case, early stopping with a higher acceptance threshold is applied. Notably, in most cases the training stops relatively early, compared to this threshold and does not continue for 300 epochs. Additionally, data-related hyperparameters like batch size, sequence window and resampling frequency are configured based on the data analysis in

Section 3.3.2. Network-related hyperparameters, such as momentum rate is adjusted with basis in the default value provided in Keras. Nevertheless, the most important initial configurations are summarised in **Table 4.1**.

Each data point is resampled to 10-hour average activity levels whereas the length of the initial input sequences are 28 data points long. This is the equivalent of observing average activity levels for approximately 12 days; $10 \times 28 = \frac{280}{24} \approx 12$.

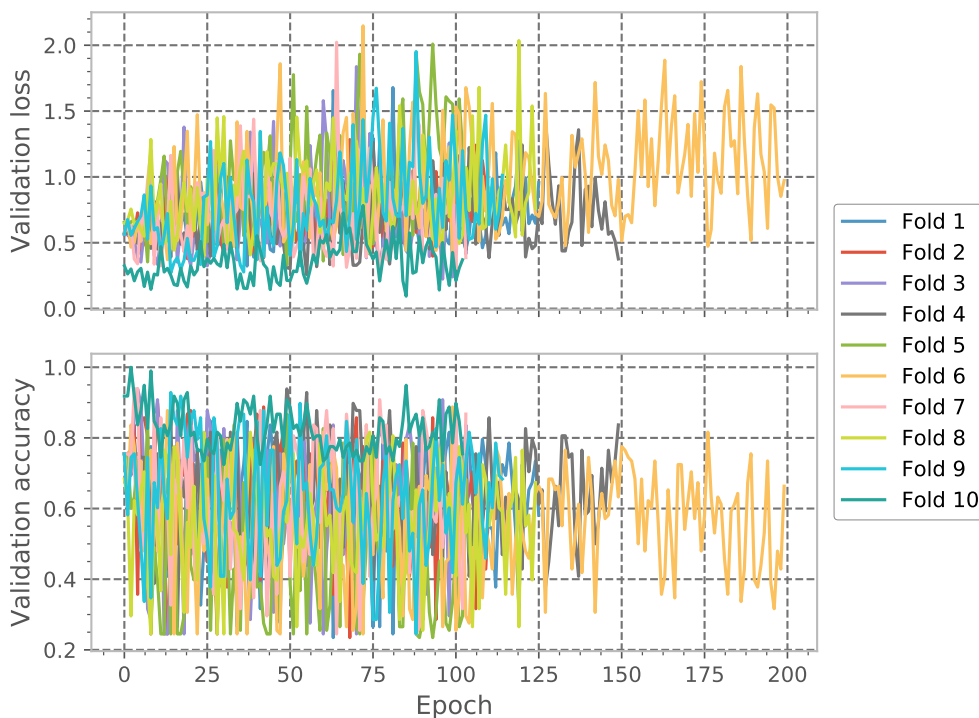


Figure 4.2: Validation history of first initial LSTM-result in the depression use case.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.760	0.857	0.578	0.760	0.766	0.238
std	0.169	0.089	0.283	0.169	0.160	0.247
min	0.500	0.737	0.179	0.500	0.514	-0.018
max	1.000	1.000	0.877	1.000	1.000	0.629

Table 4.2: Overview of weighted average metrics across all folds. The overall metric values varies, and the deviations are higher, as seen in the standard deviation but also amplified by max and min values, illustrating the larger gaps.

The first initial result is summarised in **Table 4.2** and the validation loss- and accuracy during training is illustrated in **Figure 4.2**. Overall, convergence is low and there are high fluctuations during training. Apart from these deviations, the validation accuracy decreases over time whereas an increase in loss is observed, indicating model overfitting which is observed across all folds.

Although the model overfits, another concern relates why overall deviations are higher. Hypothetically, some factors may be related to the input vectors,

such as length of the sequence window or the resampling frequency, in which temporal dependencies are not captured efficiently. Comparably, fluctuations can be related to noisy optimisation as well. For instance, batch sizes and momentum rate may affect the optimisation, similar to what was discussed in **Section 2.3.2** about backpropagation earlier. A smaller batch size contributes to more fluctuating gradients, because the batch is less representative. Additionally, increased momentum potentially results in overshooting optima in a loss landscape. To better understand this, additional experiments are performed in which the batch size is adjusted first.

I_2 : second initial result

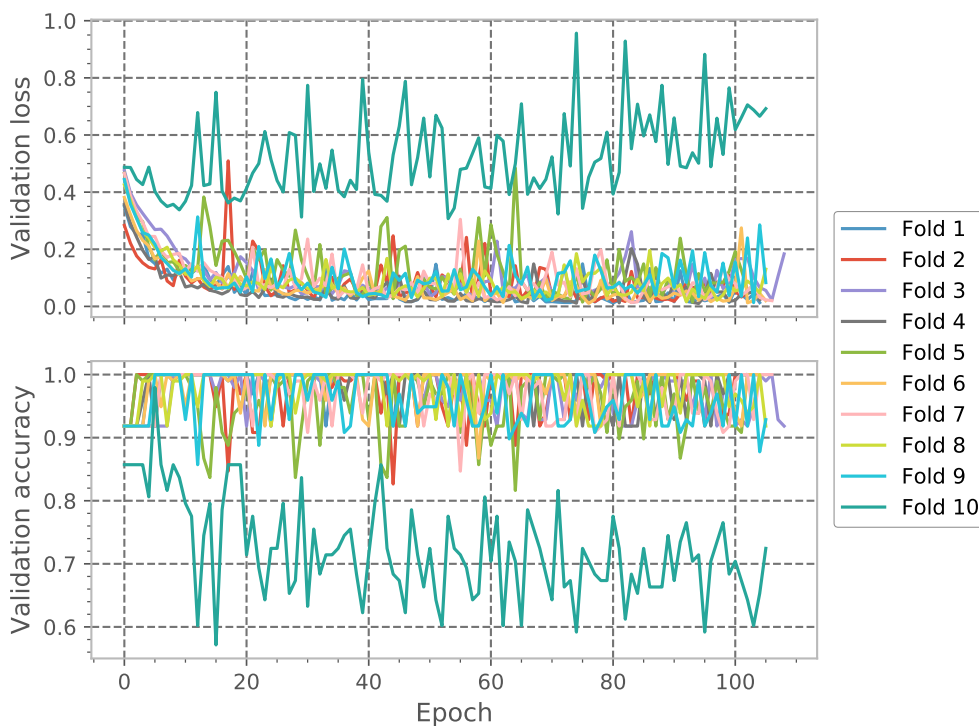


Figure 4.3: Validation history of second initial LSTM-result in the depression use case.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.783	0.815	0.539	0.783	0.780	0.238
std	0.152	0.144	0.294	0.152	0.159	0.308
min	0.519	0.501	0.074	0.519	0.492	-0.220
max	1.000	1.000	0.784	1.000	1.000	0.604

Table 4.3: Overview of weighted average metrics across all folds for second initial LSTM result. Although there are some deviations, the training history indicates that this may be due to the partitioning of the dataset.

The batch size is increased to 10, with otherwise same configurations defined in **Table 4.1**. Corresponding weighted average metrics are summarised in **Table 4.3** in

which validation history is illustrated in **Figure 4.3**.

There is a notable difference in validation history across all folds compared to the first initial experiment. Initially, MCC is more closer to zero and deviations are higher for, possibly indicating more random predictions. Additionally, with slightly higher deviations for specificity, the true negative rate, it may indicate a confusion in prediction of non-depressed patients. Specificity is also relatively lower compared to the other metrics, whereas the recall rate is higher. Evidently, classification of depressed patients seems to be more efficient although this seems not to be the case for non-depressed patients.

The overall deviations in evaluations and during training may also be explained by other factors. For instance, the 10-th validation fold seems to be an outlier partition, where the model overfits. Arguably, this could be a result of dataset partitioning, because initial shuffling of the dataset introduces a certain randomness. Some partitions may thus include activity levels that are outliers in general, for instance where depressed patients are more active than non-depressed patients or vice versa, where non-depressed patients are less active than the depressed patients.

Although this is more related to the underlying data topology and distribution. the general implication of increased batch size seems to lower the overall fluctuations in which the overall convergence seems more stable. However, it is uncertain if this is entirely true, because as shown, it may also be a side effect due to dataset partitioning. An additional iteration is to explore the other hypotheses proposed in the first initial experiment, by adjusting the resampling frequency and momentum rate.

I_3 : third initial result

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.791	0.784	0.529	0.791	0.771	0.268
std	0.216	0.223	0.364	0.216	0.240	0.398
min	0.492	0.443	0.016	0.492	0.453	-0.149
max	1.000	1.000	1.000	1.000	1.000	1.000

Table 4.4: Overview of weighted average metrics across all folds for third initial LSTM result.

Here, the resampling frequency, momentum rate and batch size is explored jointly. The series are downsampled to 12-hour average whereas batch size is increased to 15 and momentum is reduced to 0.1. From **Figure 4.4**, which illustrates the validation history, the overall reduced fluctuations and improved convergence during training is a relatively desirable outcome. There are however, minor spikes and outliers compared to earlier results which again contributes to higher deviations.

Nevertheless, most metric evaluations are relatively consistent compared to the previous result, although the precision rate has dropped the most from 0.81 to 0.78. This indicates that the relevancy in predictions is reduced. Jointly seen with the specificity rate, which is approximately the same as earlier, this can be explained by increased misclassifications of non-depressed patients, hence less relevant predictions.

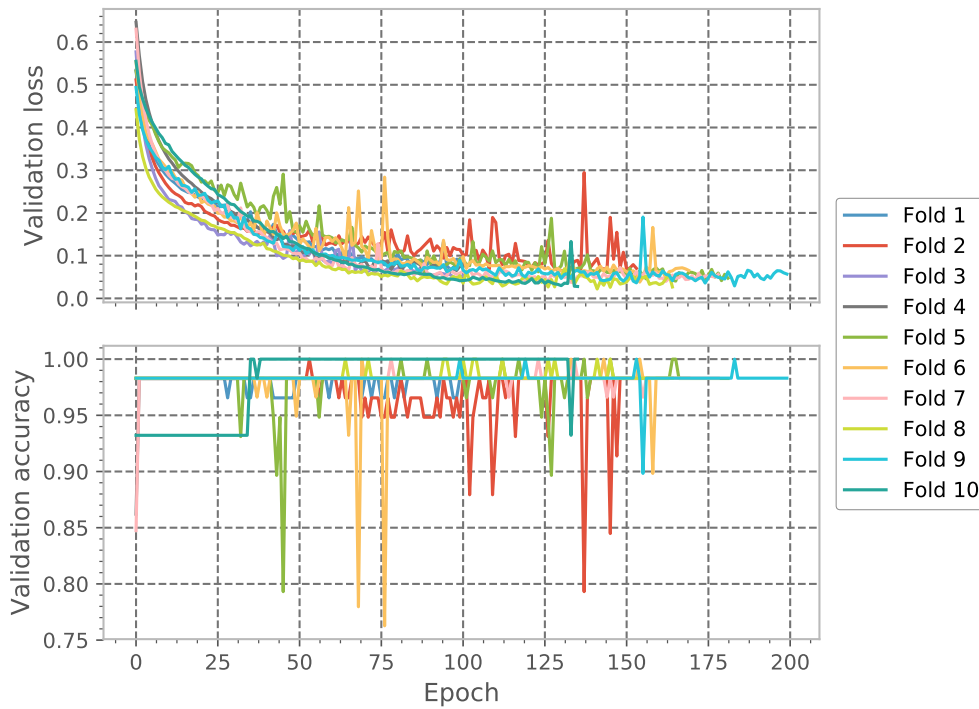


Figure 4.4: Validation history of third initial LSTM-result in the depression use case.

Conclusively, for this and the previous experiments, the batch size, sequence window length and resampling frequency have been tuned. There are notable improvements during training, although predictive performance differs between runs, where deviations are higher across predictions. A joint configuration of these hyperparameters seems to be important factors for both model convergence and performance for the LSTM as we have shown.

However, an observable outcome for all results is a decrease in specificity whereas the deviations are higher, often in response with the MCC. This outcome is most notable in the second result, where the indication may be that the network is less efficient when classifying non-depressed patients. Despite MCC being positive, the overall fluctuations and evaluation close to 0 indicates potential of improvement.

R_1 : first optimised result

The initial experiments were concerned with how certain data-related hyperparameters configurations affect model performance. One experiment was carried out to adjust the momentum rate, in which the aim was to reduce fluctuations during training. In this section, we further explore these parameters in addition to network-level parameters, where the goal is to optimise the model performance.

Similar to earlier, the data-related hyperparameters are adjusted. The initial configuration of hyperparameters is shown in **Table 4.5**, where a smaller batch of 5 is chosen, where the series is downsampled to 8-hour average. The sequence window is 30, implying the equivalent of observing average activity levels of approximately

Type	Hyperparameter	Value
Data	Batch Size	5
	Epoch	300
	Resampling Average	8H
	Sequence Window	30
Network	Dropout	0.3
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.7
	Nesterov Momentum	False
	Optimiser	sgd

Table 4.5: Hyperparameter configurations for our initial optimised result. Note that the configurations are similar to our initial results as we presented earlier, except with minor adjustments on sequence window, batch size and resampling frequency.

10 days; $8 \times 30 = \frac{240}{24} = 10$. The momentum is slightly higher similar to earlier, where fluctuations during training is expected.

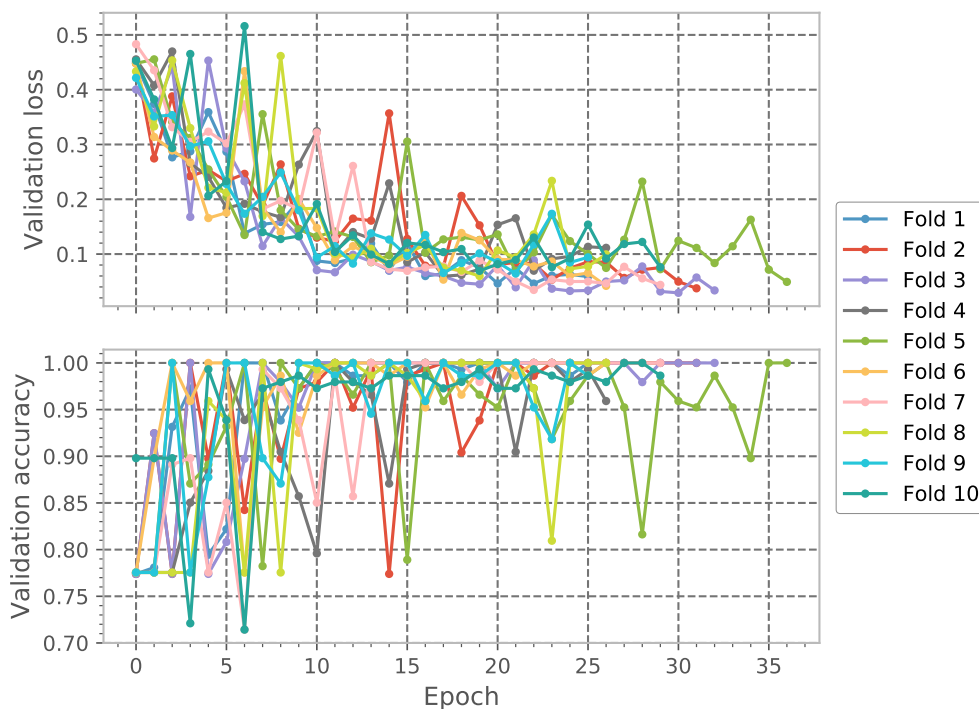


Figure 4.5: Validation history of first optimised LSTM-result in the depression use case.

The first result is summarised in **Table 4.6**, whereas the validation history during training is illustrated in **Figure 4.5**. The average results across all validation folds are promising and show improvement compared to earlier. Notably, the deviations in overall metric evaluations are far lower, but there is a recurring pattern in MCC and specificity. There are higher deviations, although there is an increase in both metrics.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.820	0.814	0.653	0.820	0.806	0.486
std	0.167	0.172	0.302	0.167	0.175	0.424
min	0.469	0.470	0.148	0.469	0.459	-0.057
max	1.000	1.000	1.000	1.000	1.000	1.000

Table 4.6: Overview of weighted average metrics across all folds for first optimised LSTM result.

The MCC rate is now 0.49, an improvement of approximately 0.2, while the specificity is 0.65, an increase of about 0.1 from initial experiments.

From **Figure 4.5**, the fluctuations are more visible initially during the training process, but the overall convergence is promising. The training process terminates after 36 epochs due to early stopping but the loss is relatively low and has already almost converged to 0. Overall, the deviation during training validation is more notably early in the process.

Nevertheless, the hyperparameter configurations are approximately the same as the initial experiments, but minor adjustments tend to result in many different outcomes. As discussed earlier in the methodology chapter, hyperparameter configuration is arguably a field of its own and the modelling problem is also highly dependent on the underlying data distribution. Thus, the variations across the presented results may be explained by this, although it may be related to other factors as well.

For instance, earlier it was discussed how these deviations may be related to particular cases where some depressed patients are more active than non-depressed patients or vice versa, where non-depressed patients are less active than the depressed patients. Evidently, this pattern is recurring throughout the experiments, either in the form of deviations in the specificity rate and MCC or during training where certain partitions in the cross-validation result in outlier predictions.

One potential solution to this problem however, is to apply discriminative features to better detect depressed patients, which is also proposed by Garcia-Ceja et al. [29] in their paper. In terms of configurable hyperparameters however, in which their combinations may result in infinitely possible solutions, the current configuration is a good basis for further optimisation.

R_2 : second optimised result

Further on, the goal is to reduce overall fluctuations during training by adjusting the momentum and learning rate of the network. With basis in the configurations defined earlier in **Table 4.5**, the learning rate is increased to 0.005 from 0.001 and momentum is reduced to 0.5 from 0.7.

Weighted averages for the second result is shown in **Table 4.7**. Compared to R_1 , there are improvements in certain metric evaluations, despite some deviations. Notably, there is an improvement in the precision rate, specificity rate and MCC. This indicates better relevant predictions in addition to classification of non-depressed patients

being improved. Additionally with an increase in MCC to 0.59 from 0.49, there is a higher agreement with the actual observations. The increase in specificity and precision suggests that the model shows the ability to better distinguish between depressed and non-depressed patients. However, there is a minor tradeoff in accuracy, recall and F_1 , implying that the overall performance may be reduced.

Anyhow, it looks like this has minor implications on overall performance, because the relevancy in predictions has increased and the deviations in validation history is smaller and consistent across all validation folds, except from one outlier partition, illustrated in **Figure 4.6**. This indicates a confusion in the underlying data distribution. As discussed earlier, the pattern is repetitive, also for this particular result, although the overall performance being relatively promising.

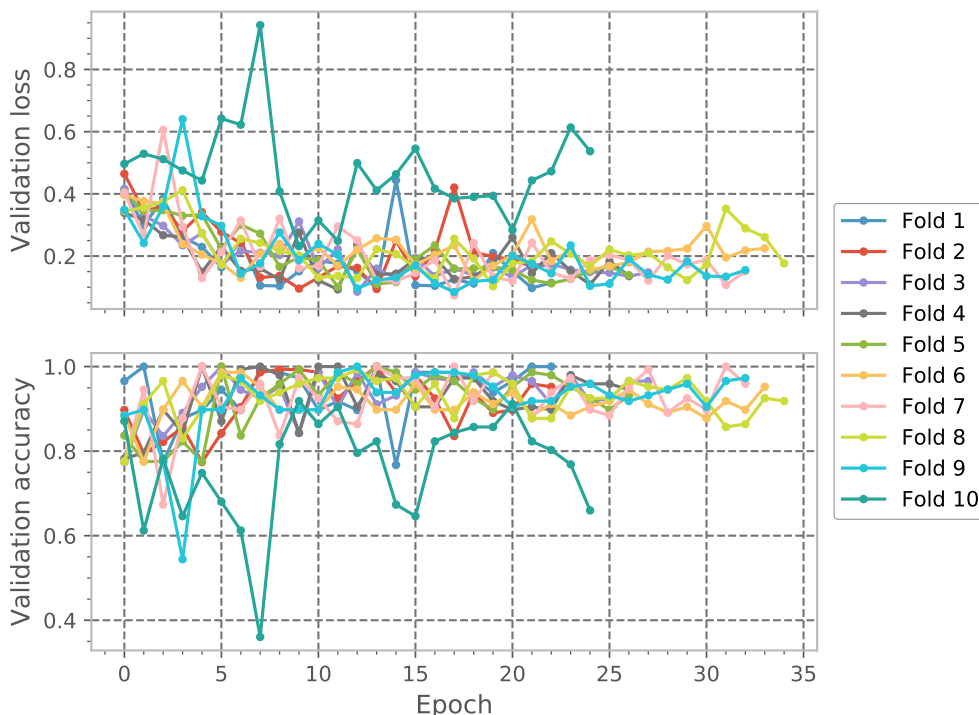


Figure 4.6: Validation history of second optimised LSTM-result in the depression use case.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.788	0.887	0.858	0.788	0.792	0.594
std	0.178	0.089	0.152	0.178	0.185	0.258
min	0.463	0.747	0.558	0.463	0.387	0.227
max	1.000	1.000	1.000	1.000	1.000	1.000

Table 4.7: Overview of weighted average metrics across all folds for second optimised LSTM result.

Overall, it is evident that applying discriminative features is a reasonable claim. The model performs the classification task well proven by the performance evaluations, implying that it has learned the most important aspects of the data distribution. However, for the outlier partitions the predictive performance is reduced. Arguably,

as discussed earlier, this is because of the underlying data distribution. Not only may it be related to how activity levels vary between patients, but activity in general may depend on other factors. This includes mood, time of day, season and weather.

Because of such conditional variability, it is hypothesised that the data distribution is more "noisy" in terms of an optimisation perspective. This can be referred to as a noisy loss landscape. The intuition is that variations in the data distribution results in a landscape with multiple optimas for the optimisation problem.

R_3 : third optimised result

Although the model in some occasions fails to capture outliers efficiently due to specific partitions, the performance gain is notable in most occasions. To further explore the intuition of a noisy landscape, the momentum rate is reduced to 0.4 from 0.5 in R_2 . The overall goal is to potentially reduce the overshooting of optima and the fluctuations in predictions.

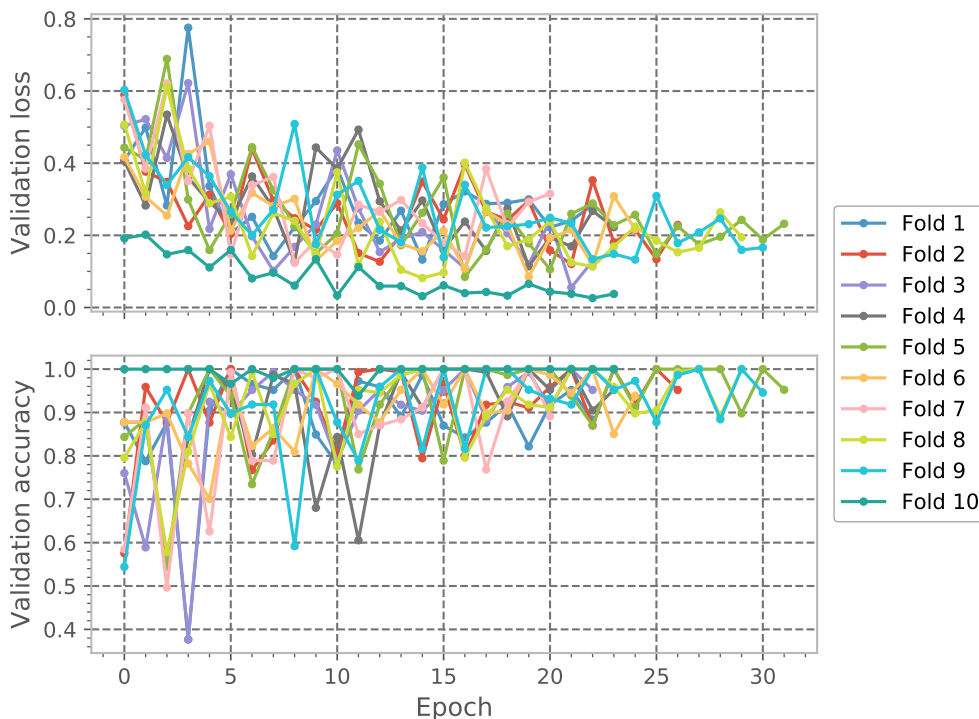


Figure 4.7: Validation history of third optimised LSTM-result in the depression use case.

As emphasised earlier, minor adjustments in certain hyperparameters results in many different outcomes, showing how sensitive hyperparameter configuration can be. From **Table 4.7**, the overall deviations are approximately the same as earlier or lower across all evaluation metrics. There is an improvement in accuracy, recall and F_1 from R_2 . From **Figure 4.7** it is also evident that predictions are relatively consistent across all validation folds. This indicates overall robustness in predictions, although a tradeoff in precision, specificity and MCC from R_2 is observed.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.818	0.858	0.789	0.818	0.823	0.504
std	0.130	0.113	0.167	0.130	0.124	0.294
min	0.613	0.672	0.539	0.613	0.634	0.000
max	1.000	1.000	0.989	1.000	1.000	0.916

Table 4.8: Overview of weighted average metrics across all folds for third optimised LSTM result.

Summary of results for LSTM

	R_1	R_2	R_3	Linear SVM
ACC	0.820 (0.167)	0.788 (0.178)	0.818 (0.130)	0.727
PREC	0.814 (0.172)	0.887 (0.089)	0.858 (0.113)	0.735
SPEC	0.653 (0.302)	0.858 (0.152)	0.789 (0.167)	0.726
REC	0.820 (0.167)	0.788 (0.178)	0.818 (0.130)	0.729
F1	0.806 (0.175)	0.792 (0.185)	0.823 (0.124)	0.724
MCC	0.486 (0.424)	0.594 (0.258)	0.504 (0.294)	0.433

Table 4.9: Summary of weighted average evaluations of all optimised results for LSTM experiments, with the standard deviation shown in parantheses. The last column is the weighted average baseline for the Linear SVM classifier [29].

Table 4.9 summarises all optimised experiments. Despite each successive experiment is not an improvement, it looks like on average, the best results are derived from R_2 followed by R_3 . For both results, the precision, specificity and recall is higher compared to R_1 , indicating more relevant predictions and better classification of depressed patients.

Also, a recurring pattern is observed throughout the experiments. The LSTM-network manages to learn the data distribution efficiently, only using about 30 to 35 epochs. In many cases fluctuations are higher during training although the training convergence indicates better optimisation. However, one observable outcome is how certain outlier partitions affects model performance. In these cases the LSTM-network fails to learn the underlying distribution well, possibly because of what was referred to as a noisy loss landscape. The intuition is how the variability in the data distribution results in multiple optimas and affects the optimisation problem.

More over, the results are compared against the Linear SVM baseline provided by Garcia-Ceja et al. [29]. From **Table 4.9** we see an improvement across all results compared to the baseline, which is promising. Although the paper provides class-level evaluations, we only present the weighted average in which the LSTM shows improvement on the Linear SVM.

4.1.3 Results for CNN

I_1 : first initial result

Type	Hyperparameter	Value
Data	Batch Size	10
	Epoch	300
	Resampling Average	12H
	Sequence Window	14
Network	Dilation Rate	1
	Dropout	0.3
	Kernel Size	10
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.1
	Nesterov Momentum	False
	Optimiser	sgd
Padding	causal	

Table 4.10: Most important hyperparameter configurations for the initial results.

Similar to earlier, the maximum training iterations is set to 300 epochs. For this initial experiment a higher acceptance threshold is applied for early stopping, although it later is adjusted for the second initial experiment and is not applied at all. The initial configurations are shown in **Table 4.10**, with CNN-specific hyperparameter configurations mainly constrained to dilation rate (dilated convolutions), kernel/filter size and padding in which a default stride of 1 is used. The hyperparameter value *causal* for the padding option is a Keras-specific term for dilated convolutions. A dilation rate greater than 1 with the option *causal* implies dilated convolutions, whereas the opposite indicate normal convolutions.

The initial batch size is set to 10 with the series downsampled to 12-hour average. A sequence window of 14 is used, which is the equivalent of 7 days average activity; $\frac{12 \times 14}{24} = \frac{168}{24} = 7$. Also, the momentum is set to 0.1 and the kernel/filter size is 10.

From **Figure 4.8**, the model overfits for all validation folds. However, loss convergence is optimal with overall minor fluctuations until 60-70 epochs, before the uncertainty in predictions is notable. Looking at the validation accuracy the overall deviations are smaller, apart from the last validation being an outlier. This recurring pattern, which we also discussed earlier in the results for LSTM, is justified as potential other factors that contribute to depression. Additionally, the randomness in dataset partitioning may also contribute to the outliers. From the validation history, it is evident that the CNN fails to capture these as well.

Despite configurations like momentum and kernel size are set randomly, the CNN-model produce results closely comparable with the LSTM and the current configurations seems to be optimal. Further on however, we aim to use dilated

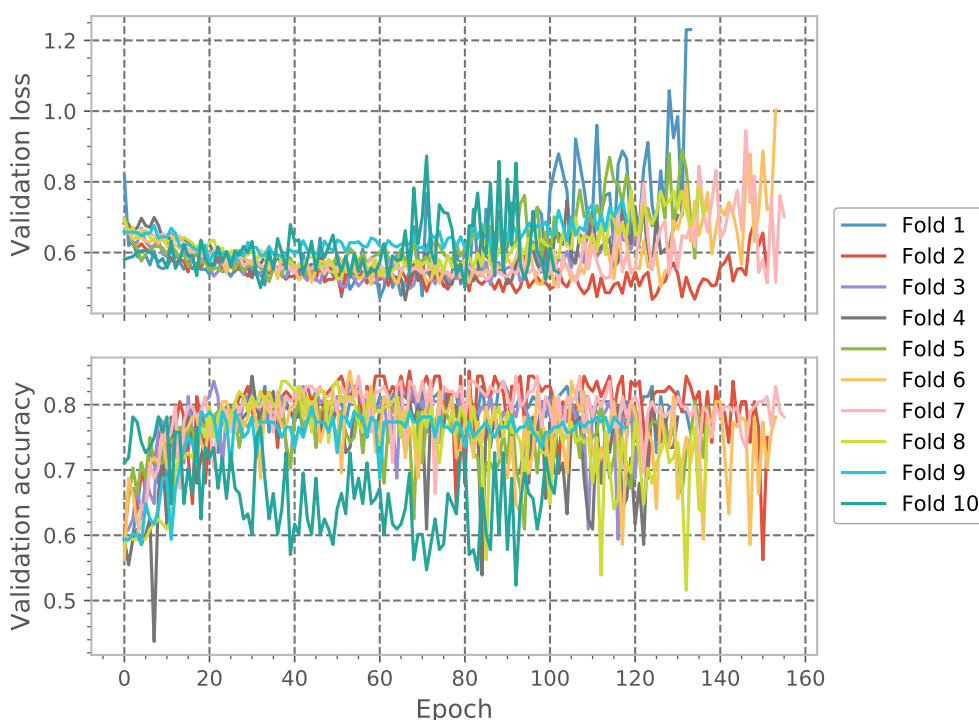


Figure 4.8: Validation history of first initial CNN-result in the depression use case.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.746	0.804	0.699	0.746	0.737	0.385
std	0.211	0.164	0.187	0.211	0.225	0.360
min	0.296	0.442	0.254	0.296	0.257	-0.258
max	1.000	1.000	0.876	1.000	1.000	0.807

Table 4.11: Overview of weighted average metrics across all folds for first initial CNN result.

convolutions as an additional experiment, where kernel size and dilation rate is explored.

I_2 : second initial result

From the previous initial result, the batch size is increased from 10 to 20, and the series are downsampled from 12-hour average to 10-hour average. A corresponding change in the sequence window length is made from 14 to 35 data points. This is the equivalent of approximately 2 weeks; $\frac{10 \times 35}{24} = \frac{350}{24} \approx 15$. Additionally, a dilation rate of 4 is used whereas the kernel size is reduced from 10 to 6 from the previous experiment. The overall size of the sliding window/filter is thus 24.

Compared to earlier, certain validation folds does not show tendencies of overfitting, although others seem to do so. The overall loss convergence is lower over time for particular validations, like the fifth and ninth validation fold. Because it is also difficult to interpret the validation history for this result, a moving average plot is

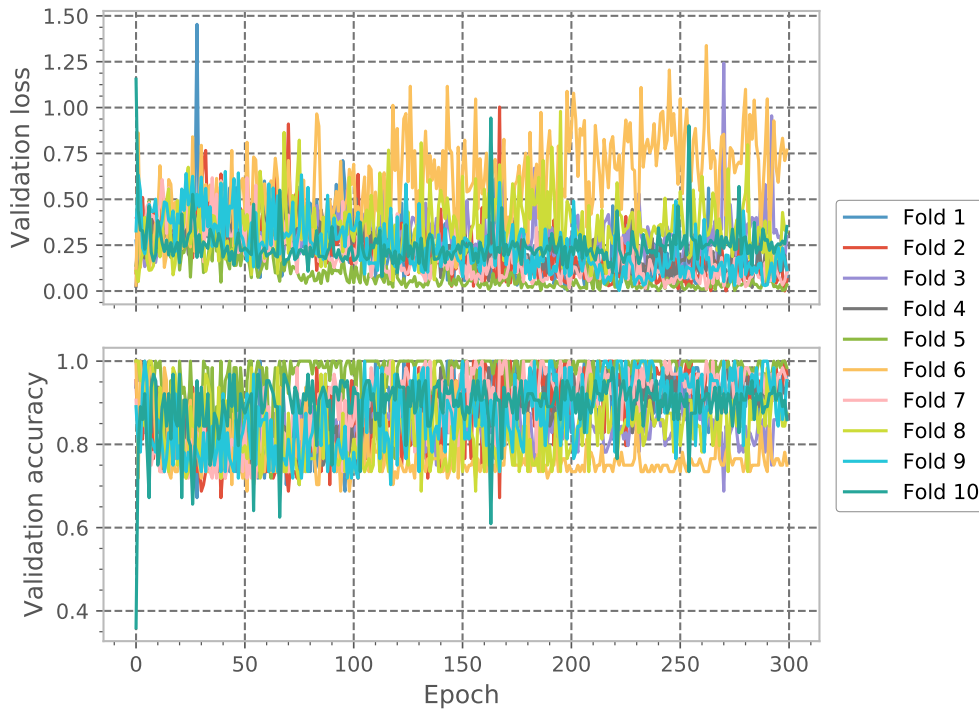


Figure 4.9: Validation history of second initial CNN-result in the depression use case.

provided in **Appendix B.1**. A corresponding confusion matrix of the best performing model is also presented in **Appendix B.2**.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.844	0.896	0.483	0.844	0.848	0.204
std	0.128	0.131	0.288	0.128	0.154	0.283
min	0.614	0.630	0.014	0.614	0.552	-0.014
max	1.000	1.000	0.898	1.000	1.000	0.838

Table 4.12: Overview of weighted average metrics across all folds for second initial CNN result.

From **Table 4.12** the specificity is lower, but the overall evaluation is promising. However, because some validation folds show tendencies of overfitting the results are not directly comparable, but gives a good indication for the current configurations.

Nevertheless, it is difficult to explain whether or not this performance gain is due to the change in kernel size and dilation rate or if it may be related to the data-related hyperparameters like batch size and sequence window, which we also tuned differently for this particular experiment. Further on, to explore this, the batch size and the length of sequence window is adjusted, while the same configurations are used for dilated convolutions. The intention is to understand how dilated convolutions potentially contributes to performance increase. Additionally, other specific hyperparameters on a network-level are explored for further optimisations.

R_1 : first optimised result

Type	Hyperparameter	Value
Data	Batch Size	15
	Epoch	300
	Resampling Average	10H
	Sequence Window	30
Network	Dilation Rate	4
	Dropout	0.3
	Kernel Size	6
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.1
	Nesterov Momentum	False
	Optimiser	sgd
Padding	causal	

Table 4.13: Most important hyperparameter configurations for the optimised results.

The base configuration for the optimised results is shown in **Table 4.13**. From the second initial result presented above, it was discussed how dilated convolutions potentially improve overall predictions. To further iterate on this, the batch size and sequence window is reduced from 20 to 15 and 35 to 30, respectively, from the previous experiment.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.820	0.867	0.635	0.820	0.819	0.345
std	0.117	0.088	0.162	0.117	0.137	0.236
min	0.557	0.713	0.344	0.557	0.510	0.000
max	0.948	1.000	0.820	0.948	0.974	0.589

Table 4.14: Overview of weighted average metrics across all folds for first optimised CNN result.

As seen in **Figure 4.10**, overall deviations are lower and the same recurring outlier partition is observed for the last validation fold. Apart from this, the training convergence is desirable, where it also stops after approximately 70 epochs in most cases. Additionally, from **Table 4.14**, the most notable deviations relates to MCC, which is 0.345 with a standard deviation of 0.236. Although the implication is an agreement in the predictions to some extent, the deviations in MCC indicates higher variations across all validations.

Also, the specificity is relatively lower indicating that the rate of correctly predicted negative classes, the non-depressed patients, is somewhat lower compared to the recall. The recall rate is closer to 1, implying that classification of depressed patients is more accurate than the classification of non-depressed patients. Similar to the experiments for LSTM, where the same behaviours were observed, it is a stronger

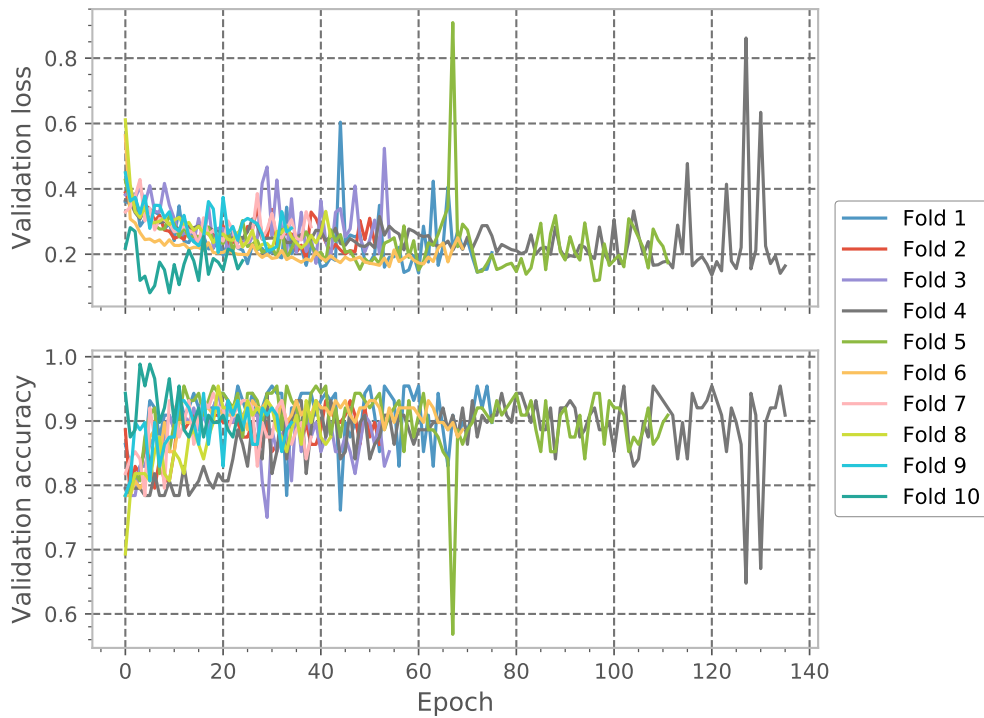


Figure 4.10: Validation history of first optimised CNN-result in the depression use case.

confirmation of the underlying data topology. It is also possibly related to the fact that depression may have different causes and there may be non-depressed patients who are not equally active as the average person.

Nevertheless, recall the motivation for this experiment in which the intention was concerned with how network-level hyperparameters such as dilation rate and kernel size yielded promising results. Notably, it is evident that the CNN performs closely well compared to the LSTM. Arguably, it may relate to how the convolution operation functions, which is similar to the gating mechanisms in LSTM.

The convolution operation on uni-dimensional data like time-series can be interpreted as a moving average. Intuitively, the kernel size is interpreted as a sliding window, essentially a subset of the whole input sequence. Hypothetically, by convolving the kernel/filter across the input sequence, the convolution operation provides a similar mechanism found in gated units. The temporal structure in subsequences is learned similarly, in which more localised patterns are captured, because the filter only looks at a subset of the sequence. We argue this is one explanation to CNN performing almost as good as the LSTM-network. Overall, with dilated convolutions and increased filter size, it looks like temporal structure is learned efficiently. Arguably this is done with less computational effort, because CNNs have localised connections whereas the LSTM has more complex gating mechanisms with increased number of parameters.

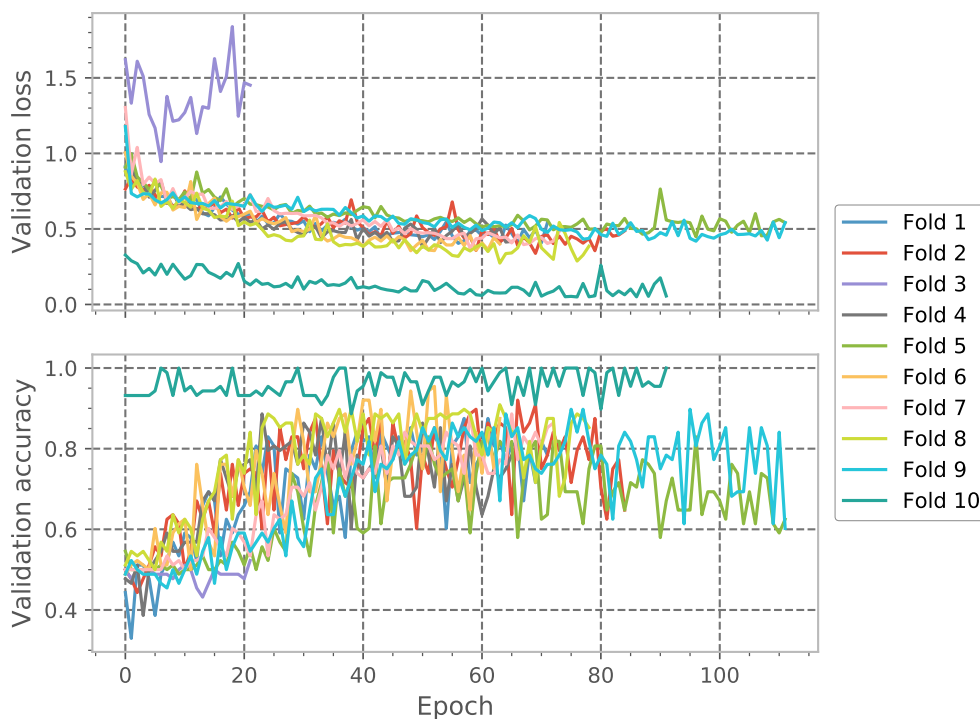


Figure 4.11: Validation history of second optimised CNN-result in the depression use case.

R_2 : second optimised result

For the second result, the momentum rate is explored in which the aim is to optimise model convergence on dilated convolutions. The momentum is increased from 0.1 to 0.3 from the initial configurations. From the resulting validation history shown in **Figure 4.11**, the overall deviations are minor. However, the loss convergence is slow and overall loss after early stopping is relatively higher compared to R_1 earlier. At a certain point after about 50 epochs in many cases, the accuracy is declining, which in fact may indicate that the increase in momentum contribute to overshooting of local optimas in the loss landscape. Lastly, similar to earlier, particular outlier partitions are observed, although it is only in one case where the model does not improve at all.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.780	0.874	0.813	0.780	0.785	0.542
std	0.202	0.116	0.226	0.202	0.219	0.364
min	0.361	0.673	0.354	0.361	0.278	-0.054
max	1.000	1.000	1.000	1.000	1.000	1.000

Table 4.15: Overview of weighted average metrics across all folds for second optimised CNN result.

Despite this, the overall weighted average indicates some improvement. This is evident, especially for the specificity rate and MCC. Evaluations have increased from 0.635 to 0.813 and 0.345 to 0.542 for both, respectively, compared to the previous experiment. As discussed in earlier results, this possibly implies that the model is

better at detecting non-depressed patients. This is further supported by the increase in MCC, showing increased agreement with the predictions and actual observations.

Although increase in momentum improves certain aspects regarding the classification task, the converged loss is higher, although overall fluctuations are lower. Whether higher loss is due to overshooting of optimas or too slow convergence is uncertain. For instance, the learning rate could be too small for this experiment, contributing to slow convergence whereas the increase in momentum could contribute to overshooting of optimas. Nevertheless, to further observe this, the momentum is reduced to 0.1 again, however with the applications of Nesterov momentum.

R_3 : third optimised result

Nesterov momentum is an optimisation technique which usually works as a corrective measure. Intuitively if the momentum rate results in overshooting an optimum from some point x_i to x_{i+1} , Nesterov momentum¹ uses a corrective measure by adjusting the optimisation step based on x_{i+1} .

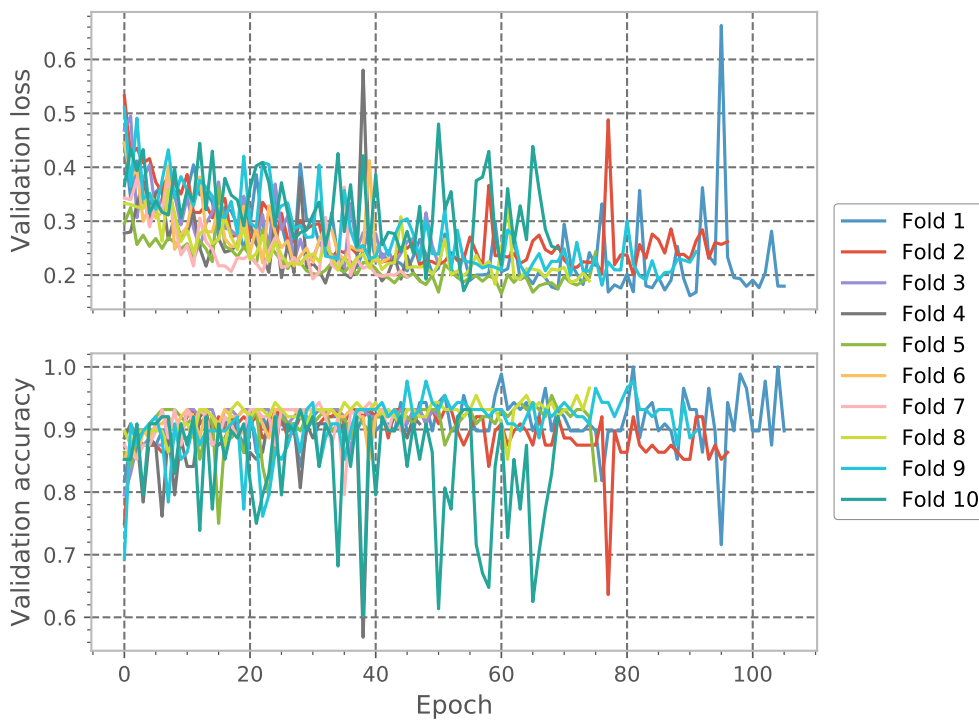


Figure 4.12: Validation history of third optimised CNN-result in the depression use case.

From **Figure 4.12**, the initial hypothesis proposed for the previous experiment, on how the model either overshoots on optima or converges too slow is evident. The corrective measure of Nesterov momentum seems to work effectively on the learning process, derived from the training convergence, which is closer to zero. The training process is more stable, because validation across all folds is approximately consistent,

¹This thesis does not cover Nesterov momentum in detail. Various optimisers exist for training neural networks, like AdaGrad and Adam which are not covered in detail. Similarly, Nesterov momentum is an extension of the original idea on momentum rate.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.816	0.844	0.629	0.816	0.811	0.411
std	0.145	0.122	0.334	0.145	0.145	0.312
min	0.567	0.576	0.052	0.567	0.558	-0.105
max	0.969	0.979	0.998	0.969	0.972	0.803

Table 4.16: Overview of weighted average metrics across all folds for third optimised CNN result.

although there are some spikes and deviations during training. Also the specificity, jointly with the MCC, is now lower compared to the previous experiment. As emphasised throughout the experiments however, this may as well be related to the data partitioning and underlying data distribution, although no particular outliers are observed during training.

Nevertheless, it seems like metric evaluations for the test partitions varies across results, but training process for the CNN is arguably an improvement for this particular experiment. Although this can not directly be explained by the application of Nesterov momentum one approach can be to increase the momentum again, while applying Nesterov. By doing this, it will be more evident whether or not Nesterov momentum actually is an improvement for the CNN.

R_4 : fourth optimised result

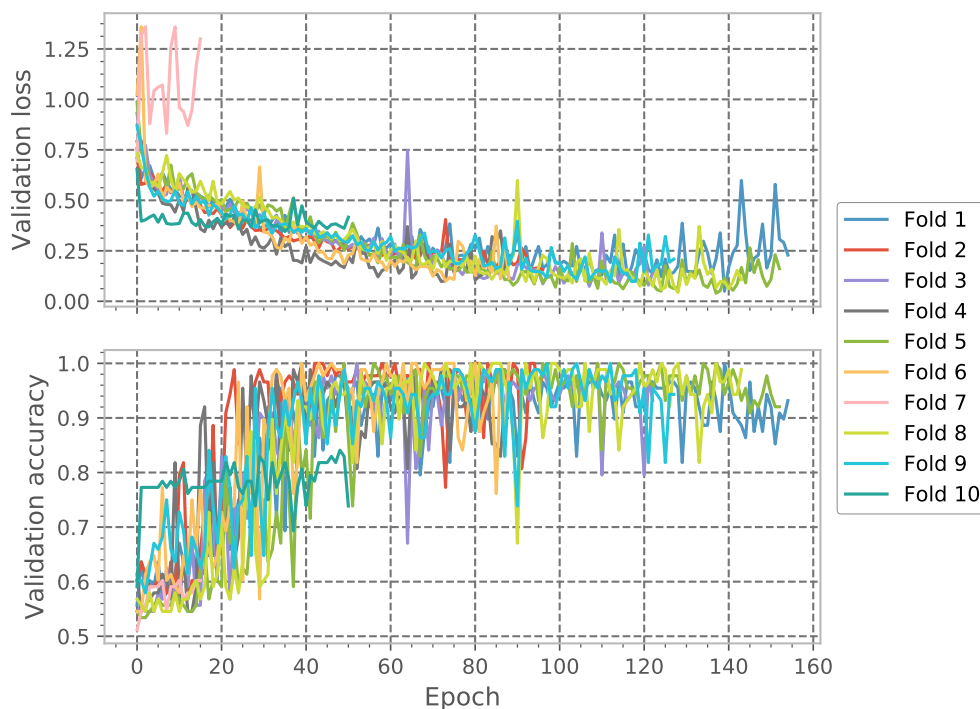


Figure 4.13: Validation history of fourth optimised CNN-result in the depression use case.

For the last experiment, the momentum rate is increased to 0.3 again, adjusted from 0.1 from R_3 earlier, in which the validation history is shown in **Figure 4.13**. Arguably, there is a reasonable improvement in terms of robustness. The overall loss converges even closer to zero, whereas this is consistent across all validations with minor deviations during training. This could be an indication of efficient loss optimisation with the specified hyperparameters, possibly also a positive effect of Nesterov momentum.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.801	0.813	0.711	0.801	0.791	0.422
std	0.145	0.209	0.288	0.145	0.187	0.366
min	0.536	0.342	0.210	0.536	0.417	-0.211
max	0.990	0.991	0.999	0.990	0.990	0.952

Table 4.17: Overview of weighted average metrics across all folds for fourth optimised CNN result.

From the weighted average evaluations of all test folds in **Table 4.17** the specificity rate has increased to 0.711 from the R_3 , although still lower than 0.813 from R_2 . Similarly, the MCC is slightly increased to 0.422, although the current best is 0.542 from R_2 .

In general, it looks like there are particular evaluation metrics that fluctuate more. A recurring pattern is the specificity rate and MCC which seems to have higher deviations, compared to the other evaluations. Although the last experiment is not the most optimised result, the overall performance and training process seems more robust. The loss across all validations is minimal, and metric evaluations are approximately the same, despite slightly reduced compared to earlier experiments.

Summary of results for CNN

	R_1	R_2	R_3	R_4	Linear SVM
ACC	0.820 (0.117)	0.780 (0.202)	0.816 (0.145)	0.801 (0.145)	0.727
PREC	0.867 (0.088)	0.874 (0.116)	0.844 (0.122)	0.813 (0.209)	0.735
SPEC	0.635 (0.162)	0.813 (0.226)	0.629 (0.334)	0.711 (0.288)	0.726
REC	0.820 (0.117)	0.780 (0.202)	0.816 (0.145)	0.801 (0.145)	0.729
F1	0.819 (0.137)	0.785 (0.219)	0.811 (0.145)	0.791 (0.187)	0.724
MCC	0.345 (0.236)	0.542 (0.364)	0.411 (0.312)	0.422 (0.366)	0.433

Table 4.18: Summary of weighted average evaluations for all optimised results. Standard deviations are shown in parantheses whereas the last column is the weighted average baseline for the Linear SVM classifier [29].

Table 4.18 summarises all optimised experiments. Similar to the experiments for LSTM, each successive experiment is not necessarily an improvement. However, on a average level it looks like the best results can be derived from R_2 , where the specificity

rate and MCC is higher. Also, accuracy, recall and F_1 are slightly less than R_1 , which is a close second in terms of performance.

Notably, throughout the experiments, total used training iterations in learning the data distribution varies from slightly below 100 epochs to about 140 epochs. More over, the fluctuations during training are small, derived from the loss convergence for all optimised results. Additionally, we experience the CNN to be more sensitive to hyperparameter configurations compared to the LSTM, which required less effort.

Conclusively, compared to the Linear SVM baseline [29], the CNN also shows promising results similar to the LSTM. There are improvements across all experiments as well, although the MCC improvement is minor. More over, the temporal models shows improvement on the provided baseline. We aim to further discuss the differences between both in the following section, emphasising that the overall goal is to answer whether CNNs are efficiently applicable for time-series classification.

4.1.4 A comparison on LSTM and CNN

This section compares both models and discuss the experiments in context of depression detection in more detail. Generally, the performance of both models are promising for time-series classification and especially in terms of depression detection. Both our models show improvement across experiments on the Linear SVM baseline provided Garcia-Ceja et al. [29], which is considered the best performing baseline model.

The weighted average metrics for the cross-validation was discussed for each model earlier. Evidently, the results suggest that performance between both models varies. Number of iterations used to learn the data distribution, convergence in general and predictive performance is different for both. Across all experiments however, the deviations in predictions are minor. CNN is arguably comparable to the LSTM, in which the latter seems to perform most optimal, albeit with marginal differences. Further on, three factors are discussed where CNN and LSTM is compared. This includes training time, impact of the underlying data distribution and model architecture.

On model training time

Figure 4.14 illustrates the total training time in seconds for all experiments. The experience when training the CNN-model was most notable, as it used far less seconds than the LSTM. Initially, this motivated for many experimental configurations. Although time used in seconds varies between both models, it is important that both models are compared on equal terms, which is not necessarily the case in our experiments². Different configurations on hyperparameter settings

²Total used seconds may not be an accurate indicator of model complexity. Another measure would be comparing total optimisable parameters for the models. Recall this from the methodology chapter, where total parameters for our LSTM is 193 052 and for CNN it is 102 002. The training varies depending on runs and configurations. With the same number of hidden layers and units in each layer for all experiments, total number of parameters are constant for both our models.

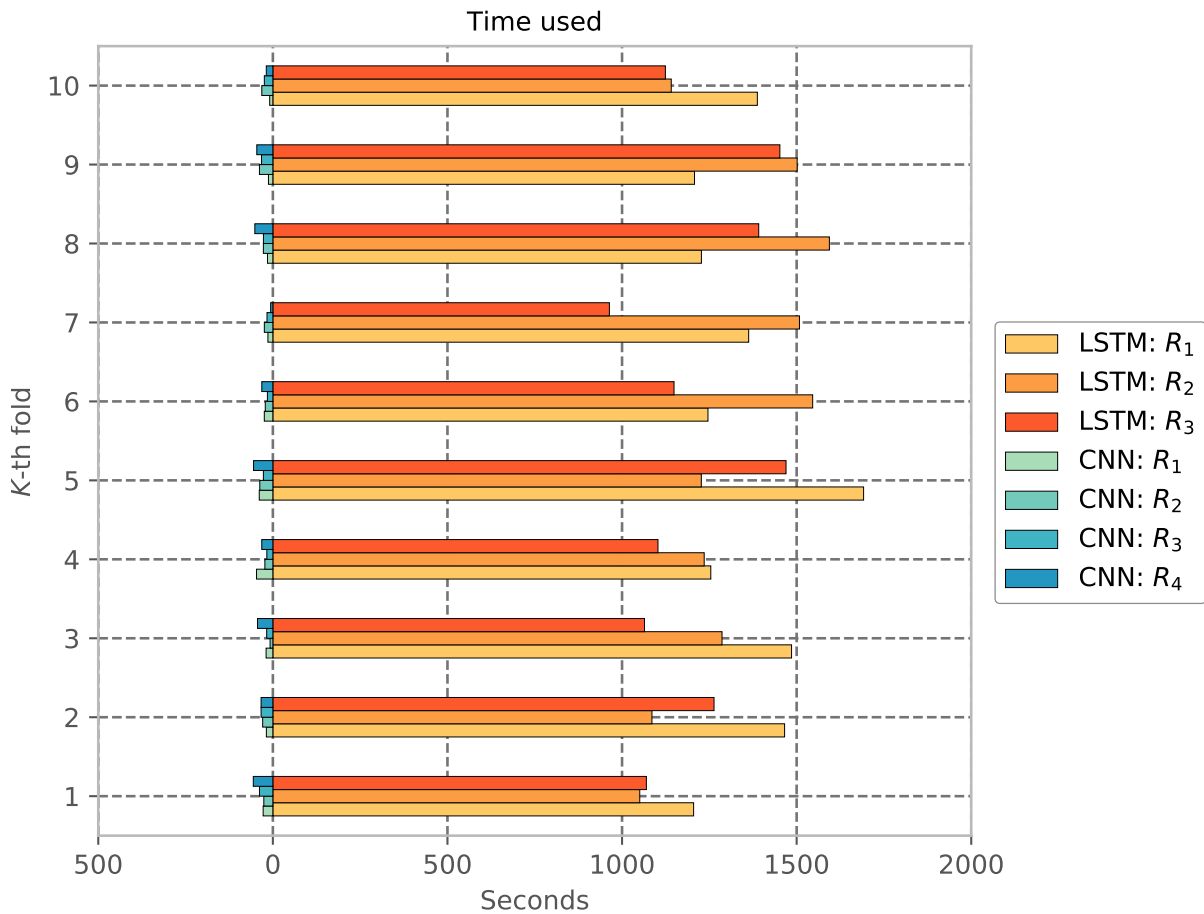


Figure 4.14: Total training time in seconds for each of the models across all results. Notably, the training time for LSTM is higher compared to CNN.

like the sequence window and resampling frequency may result in larger differences in absolute training times.

Despite this however, we emphasise that overall experience when training the models is indeed longer for the LSTM, and the figure illustrates this to an extent. The average training time for the LSTM range from approximately 1100 seconds to 1200 seconds for each fold, compared to the CNN which is faster with average times of 20 to 40 seconds, which is approximately 30 to 40 times faster.

Although we discuss total seconds used, another interesting aspect being more comparable, jointly with total seconds used, is number of training iterations before early stopping. For most LSTM experiments, the training converges already after 25-30 epochs and terminates more often around the 30-epoch mark. The CNN however, uses more epochs in learning the data distribution, closer to 100-150 epochs. Notably, the LSTM is capturing the underlying distribution 3 to 4 times more effectively, which is more comparable to absolute training time used.

Conclusively, CNN uses far less time for training in terms of seconds, more closer to 30 to 40 times faster. However, this depends on configurations and is only a relative measure. Interestingly, we see the opposite when observing the training iterations.

Despite using more time in terms of seconds, the LSTM-network uses far less training iterations compared to CNN when learning the data distribution.

On predictive performance

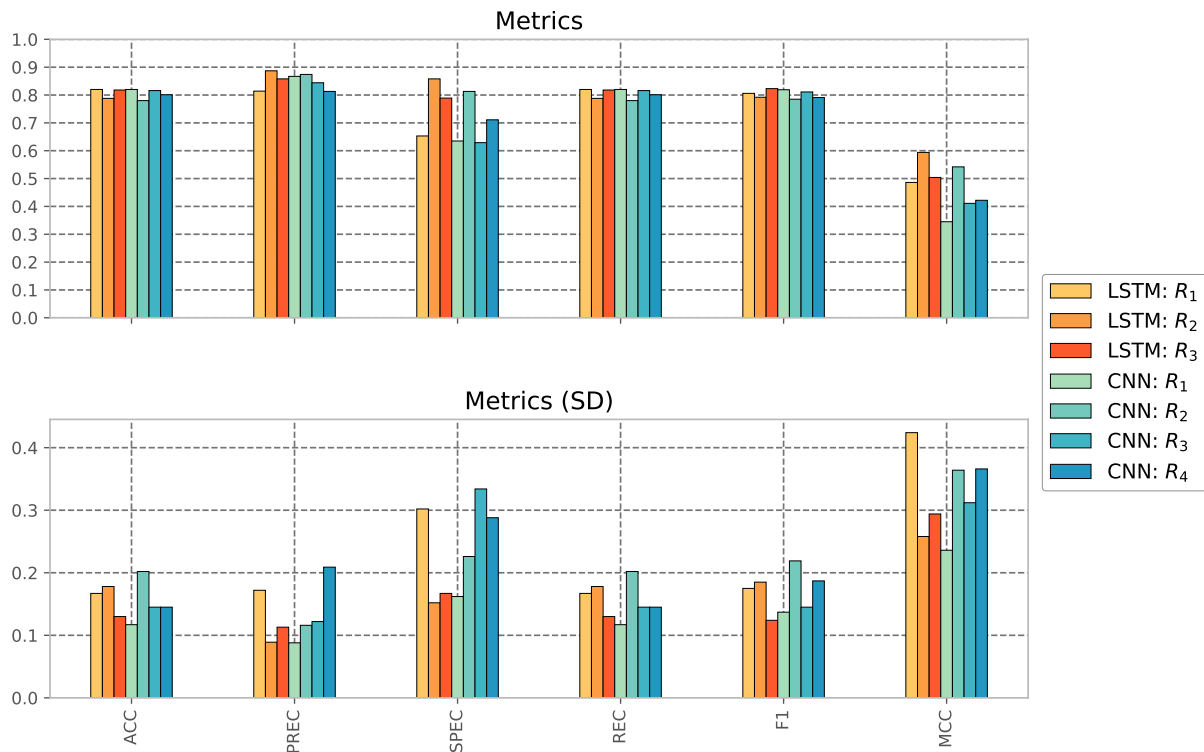


Figure 4.15: The figure shows a summary overview of the weighted average for all results presented earlier for all our models. The top figure shows the evaluation metrics where each bar represents one experiment. The bottom figure shows the standard deviations, and thus, the variations in the cross-validation. From this figure, the deviations across validations is more notable for both models, especially for the MCC and specificity.

A visual representation of the weighted averages are shown in **Figure 4.15**, which summarises evaluations for both models and all experiments. Overall differences in predictive performance for both models are minor. However, the LSTM shows more optimal performance, albeit with marginal differences. Additionally, the deviations on specificity and MCC is notable for both, illustrated by the black error bars.

In most experiments, model optimisation is often the primary concern. However, this is dependent on infinitely possible combinations of hyperparameters. Some results vary in terms of metrics and in many cases, succeeding results with different configurations are not necessarily an improvement on the previous. Despite this, the aim is to converge towards optimised models that are robust, wherein the presented metrics indicate overall performance for the classification task. Hence, the last experiments are not often those with the highest metrics, but across all results they illustrate the best performing models.

In many cases, both models show reduced capability in terms of classification,

especially in the case of non-depressed patients. This is derived by observing the variations in specificity, the true negative rate, which measures the negative class. Arguably, this is important because it implies that there is an increased number of false positives, where non-depressed patients are classified as depressed. However, more important is it to note that this confusion can better be understood by including class-level evaluations, like Garcia-Ceja et al. [29] provides in their baseline, in addition to weighted average evaluations.

Nevertheless, across all results the average specificity varies most, compared to recall which stabilises around 0.8 and in essence is a good indication of efficient classification of depressed patients. Additionally, in many cases the models were often subject to outliers in the data partitions as we saw a recurring pattern in many of our results, which contributed to variations in specificity and MCC.

This problem is true for both models, and as discussed, it may be related to confusion regarding the underlying data distribution. Also, similar to what has been proposed earlier, a solution to this problem is the addition of descriptive/discriminative input features, discussed by Garcia-Ceja et al. [29]. Comparably, as mentioned in the experiment overview, only activity mean and standard deviation is used as input features. The paper also discusses the inclusion of percentage of events with no activity, which is not included. Whether it may have affected the performance of our models is uncertain but the overall indication is that with more specific handcrafted features, the underlying distribution of the data can potentially be learned more effectively.

On data topology and distribution

Throughout the discussion and experiments it is discussed how the data distribution and particular outliers in the data partitions have resulted in different outcomes. More often, the validation history shows at worst an overfitting due to these outliers, but in most cases the training simply falls short without any improvement. We experience that neither of the models generalised well to capture these outliers efficiently, and often referred to how this relates to what was described as the loss landscape and data topology.

With this, the intuition is that the underlying distribution of data on activity levels may be ambiguous depending on patients. For instance, activity levels may also be related to lifestyle choices. Certain factors were discussed, like how some non-depressed patients may still be less active than usual or other depressed patients being more active than the non-depressed patients.

This ambiguity may also be related to the problem of multimodality in data distributions, which is concerned with distributions that have multiple "peaks", known as modes. Hence, in terms of an optimisation perspective, multiple modes may result in different topologies of the data distribution. We believe this affects the optimisations because such distributions have multiple optima and contribute to a different loss landscape, which affects the model learning.

The problem of multimodality was implied earlier in the experiments. Arguably, it is believed this is present in the activity level distribution for non-depressed patients.

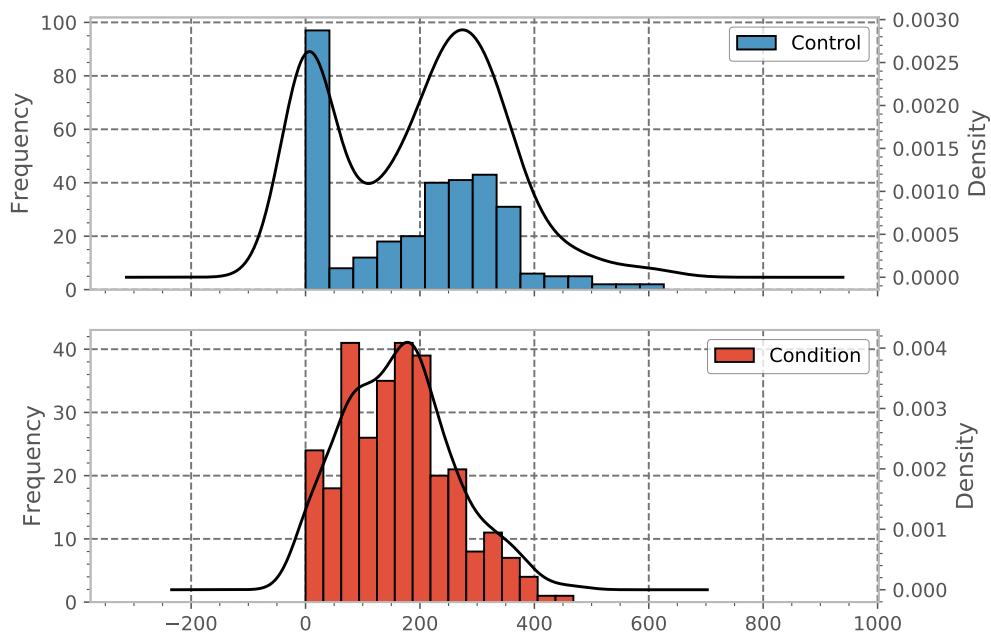


Figure 4.16: Histogram of activity levels for both depressed and non-depressed patients, with the dashed black line representing the kernel density estimation (KDE). The X-axis represents daily average activity levels, averaged across all patients whereas the Y-axis on the left side is the frequency within each interval of 15 bins. The Y-axis on the right side represents the KDE density/estimate. We use 15 bins for the histogram and Gaussian kernels are used for KDE, the latter being default in Pandas which we used to generate the figure.

More specifically, because variations in specificity were often observed, which was further explained by outliers during training history.

Figure 4.16 emphasises this problem, where distributions of activity levels for both depressed and non-depressed patients are illustrated through histograms³ and kernel density estimations (KDE)⁴[18]. Both methods gives an indication of the underlying data distribution, where the histograms show a discrete representation and KDEs show a continuous representation of the same distribution. Nevertheless, the multimodality in activity levels for non-depressed patients is evident, and confirms the hypothesis and discussions on data topology and the outliers. Both models fail to capture these in our case, due to this problem of multimodality. This shows the importance of input data for the models, which can possibly be solved by including additional features, extending the sequence window to capture longer temporal structure or include more data from additional patients to include for the variations in activity levels.

³A histogram is a bar plot where each bar represents the frequency of observations within an interval. An interval is a range of values within the set of all possible values and is determined by configuring how many "bins" the set of all possible values should be separated (intervalled) in.

⁴Kernel density estimation (KDE) is a way of estimating underlying continuous probability distributions. It uses estimator functions that looks at a particular value x and number of neighbouring observations, defined by what is known as the bandwidth. The neighbour count is an estimation of the probability of observing x , thus higher count equals higher probability.

On model architecture

Both model architectures are constrained to 4 hidden layers, hence the networks are sparse in terms of depth. Whether either of our models would perform better with deeper layers for this classification task is difficult to prove without further experiments.

However, as argued above and in earlier discussions, both models perform the classification task equally well with marginal differences. Training time was discussed, in which the CNN uses shorter time in terms of seconds but also longer time to learn the data distribution in terms of epochs. Moreover, recall that we want to answer the primary research question *How do CNNs compare to LSTMs for time-series classification?*. An important aspect in answering this question lies within the model architecture and the structural components.

The more complex gating architecture of the LSTM was designed to overcome a problem of capturing long-term temporal dependencies. We experience promising performance compared to the Linear SVM baseline provided by Garcia-Ceja et al. [29]. Notably, as shown earlier, this complexity can be explained by seconds used for training LSTM and also the number of trainable parameters in the model, although it is more efficient than CNN in learning the data distribution. Comparably, similar gating mechanisms are not present in CNN, although it achieves very similar results.

Throughout the experiments however, it was experienced how the kernel size and dilated convolutions for CNN affected performance positively, similar to how the sequence window and resampling frequency influence the LSTM. This was also mentioned earlier in the first optimised result R_1 , on how kernels/filters in CNNs may function as an equivalent to the gating mechanism in LSTM. Because the convolution operation works as a sliding window, the kernel size arguably contributes to learning localised patterns in longer sequences by providing this similar mechanism.

Combining this mechanism with dilated convolutions, it is believed that a more generalising pattern is provided, because the sliding window increases, hence widening the spatial field on the input. Arguably, this imitates a behaviour of observing longer timespans. Whether this assertion holds is difficult to say, but our results suggest that dilated convolutions are efficiently applicable. This is further supported by Borovykh, Bohte, and Oosterlee [12], where they use dilated convolutions. More over, they also show how temporal dependencies are captured more efficiently, without a need for long historical series.

Despite this curiosity on kernel size in CNN and sequence window in LSTM, we see that both models respond differently on resampled series. Our analysis earlier implied that resampling to average activity levels could help capturing temporal dependencies well. Although LSTM-units are designed for capturing long-term dependencies efficiently, we used relatively shorter sequence windows of around 30, because each data point represents the average activity level over a given timespan. One interesting aspect of this is whether simple RNN-units would be suitable with the same approach and shorter resampled sequences. Although extensive experiments with longer sequences without resampling were not performed, keeping minutely

frequency would require a window of $60 \times 24 = 1440$ data points in a sequence window to capture a 24-hour timespan. How both models respond to sequence windows of longer length is uncertain, and it is emphasised that further experiments can be carried out to explore this.

4.1.5 Summary of experiments

Conclusively, the CNN-model seems to be more sensitive to hyperparameter configurations. However, in most experiments, it shows great potential for time series classification, compared to the LSTM. Furthermore, the classification metrics are very similar for both models. Although it is difficult to notice the differences in variations, it looks like the variations across experiments are very similar as well. Moreover, the overall experience when training both models varies. The CNN-model is experienced as faster to train but uses longer time in learning the data distribution. On the other hand, the LSTM is relatively robust and uses a shorter time to achieve similar results, although it takes longer time to train.

4.2 U_2 : Energy prediction

4.2.1 Experiment overview

In this section, we discuss the experiment narrative and choices made as part of the experiments. Most importantly, due to practical constraints regarding the EV dataset, the performed experiments are limited, only showcasing the improvement potential.

Overview of narrative

From the analysis of the EV dataset, it is evident that charging and driving patterns are explained best through relative battery capacity and odometer readings. Moreover, by observing the temporal patterns over longer timespans and a daily basis, the cyclic patterns in day-to-day usage can be explained well.

First, the initial experiments showed little or no potential improvement on the classification task itself. Arguably, with a larger dataset and more representative number of EVs, there is an improvement potential in the classification task. Many additional experiments that are not presented here can be found in **Appendix B.3** and **Appendix B.4**. More importantly, we focus on exploring how the dataset affects the models and how different hyperparameters contributes to the outcome. Hence, the presented results described in **Figure 4.17** reflect a subset of the total experiments carried out, but also showcase the potential application area for this use case.

Overview of experiment preparation

The EV dataset contains hourly observations, and as described in detail about the dataset earlier, cars have been added incrementally over time since October 2018.

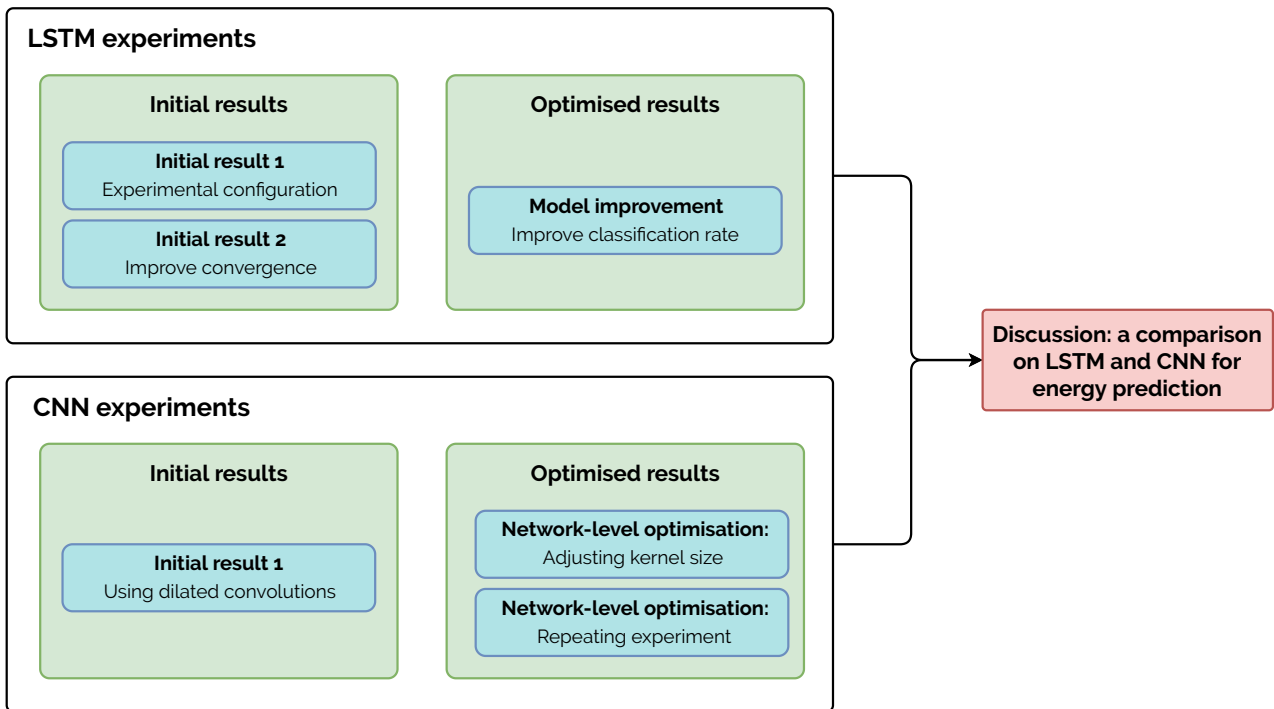


Figure 4.17: General level experiment overview for EV energy classification experiments. There are 2 initial experiments and 1 optimised experiment for LSTM, whereas 1 initial experiment and 1 optimised experiment is performed for CNN, in which the latter is ran twice.

Although the collection happens on an hourly basis, there are often periods with missing data as well. With some missing periods spanning multiple days or weeks, two approaches were considered to overcome this problem.

First, new data points can be generated, sampled from the existing data distribution. In one way, this is beneficial and increases the data volume, although working with synthetic data may not be as representative as the original dataset, because of implied noise. The chosen approach, however, is to focus on predicting daily usage with the existing data. Although the number of data points is smaller upon aggregation, the dataset is more representative and also gives a more general-level indication on consumption patterns.

Further on, a second concern is related to feature selection. The analysis in the previous section suggested that percentage battery capacity and driving distances describe the day-to-day usage well. Although the EV dataset includes features like temperature and location, the feature selection is constrained to only battery capacity and driving distances. Chosen input variables include used battery capacity,

deviational measures in battery capacity, like mean and standard deviation, and lastly the driven distance, which are all normalised to the same scale.

4.2.2 Results for LSTM

I_1 : first initial result

Type	Hyperparameter	Value
Data	Batch Size	10
	Epoch	300
	Resampling Average	D
	Sequence Window	30
Network	Dropout	0.4
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.1
	Nesterov Momentum	False
	Optimiser	nadam

Table 4.19: Most important hyperparameter configurations for the initial results.

Table 4.19 shows the initial hyperparameters, configured based on the data analysis. By default, as emphasised in the experiment design, the resampling average is constant and set to a daily average. Moreover, to capture the temporal dependencies in daily driving patterns, the sequence window is set to 30 days.

Most importantly, however, the LSTM has shown to be sensitive to multiple configurations throughout the experiments. Although not presented, the common experience has been a model subject to overfitting and inefficient convergence. In many cases, the model converges very slowly, and in some cases, not at all. As illustrated in **Figure 4.18**, this is most evident for the first initial experiment, where the model overfits for three folds and does not converge optimally otherwise.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.558	0.430	0.463	0.558	0.474	0.021
std	0.143	0.121	0.179	0.143	0.124	0.051
min	0.312	0.206	0.271	0.312	0.283	-0.049
max	0.729	0.638	0.780	0.729	0.645	0.105

Table 4.20: Overview of weighted average metrics across all folds for first initial LSTM result.

Although there may be various explanations for the problem of convergence, it could arguably be related to two aspects. First, the optimisation itself could be an explanation and second, the batch size can be less representative, resulting in lousy convergence. To further explore this hypothesis, the batch size and optimisation-

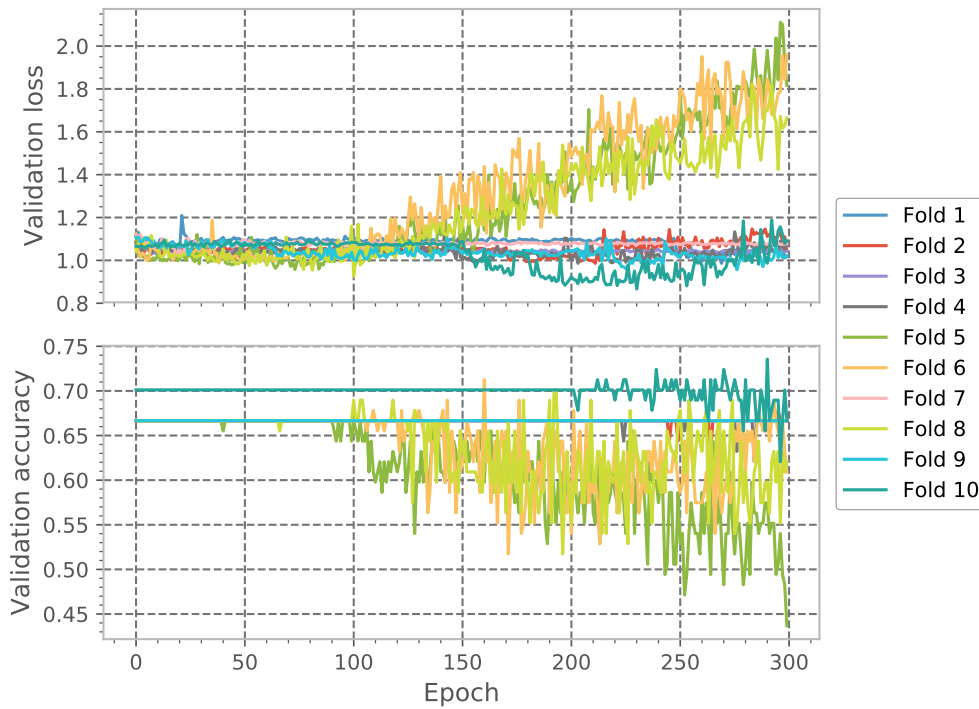


Figure 4.18: Validation history of first initial LSTM-result in the energy use case.

related parameters are tuned. More specifically, the momentum is adjusted, while the optimiser is changed and Nesterov momentum is applied.

I_2 : second initial result

For the second initial result, the momentum rate increases from 0.3 from 0.1, while the SGD-optimiser is used along with Nesterov momentum. Additionally, we reduce the batch size. Hypothetically, the overall convergence should be faster with the application of momentum, and the optimisation itself will potentially be noisier in terms of the batch size. However, as seen from **Figure 4.19** this is partially the case.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.614	0.397	0.386	0.614	0.478	0.0
std	0.146	0.162	0.146	0.146	0.169	0.0
min	0.312	0.098	0.247	0.312	0.149	0.0
max	0.753	0.566	0.688	0.753	0.646	0.0

Table 4.21: Overview of weighted average metrics across all folds for second initial LSTM result.

From the validation history, the convergence in loss is desirable, indicating efficiency in optimisation. However, the convergence overall is constant after about 20 epochs. When comparing this against the validation accuracy, the same pattern is observed, and the predictive performance of the model is mostly constant throughout the whole

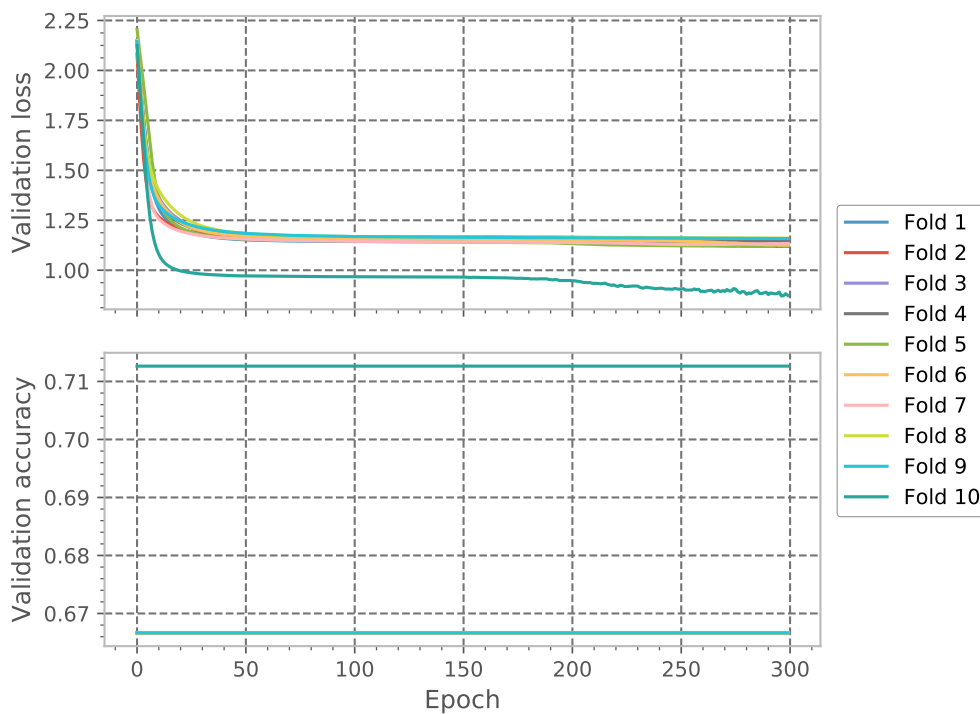


Figure 4.19: Validation history of second initial LSTM-result in the energy use case.

training process. Additionally, from **Table 4.21**, it is evident that the predictions may be more random as well, indicated by the MCC being 0.

Despite the convergence in the loss being reasonable, the classification rate of the model does not improve. One explanation to this may be related to uncertainties in the underlying data distribution which affects the classification performance. However, it could as well be related to the stochastic nature of hyperparameter settings.

Nevertheless, because of time, resource and dataset constraints, extensive optimisation of hyperparameters was not performed, although many experiments were carried out⁵. In the following section, we discuss one of the optimal outcomes.

R_1 : optimised result

While many experiments were carried out with different hyperparameters, the most optimal outcome uses the configurations defined in **Table 4.22**. The validation history in **Figure 4.20** indicates a good convergence in loss, but similar to earlier, the classification rate of the model is approximately constant. As discussed, there may be multiple reasons for this. However, the experience throughout the experiments indicates that it may be related to the dataset because of the same pattern emerging for different hyperparameters as well.

Moreover, as shown in **Table 4.23**, the weighted average MCC is negative, closer to zero, which indicates almost more randomness in predictions. The standard

⁵Validation history for additional LSTM experiments not presented here are found in **Appendix B.3**

Type	Hyperparameter	Value
Data	Batch Size	15
	Epoch	300
	Resampling Average	D
	Sequence Window	21
Network	Dropout	0.4
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.1
	Nesterov Momentum	False
	Optimiser	nadam

Table 4.22: Most important hyperparameter configurations for the optimised result.

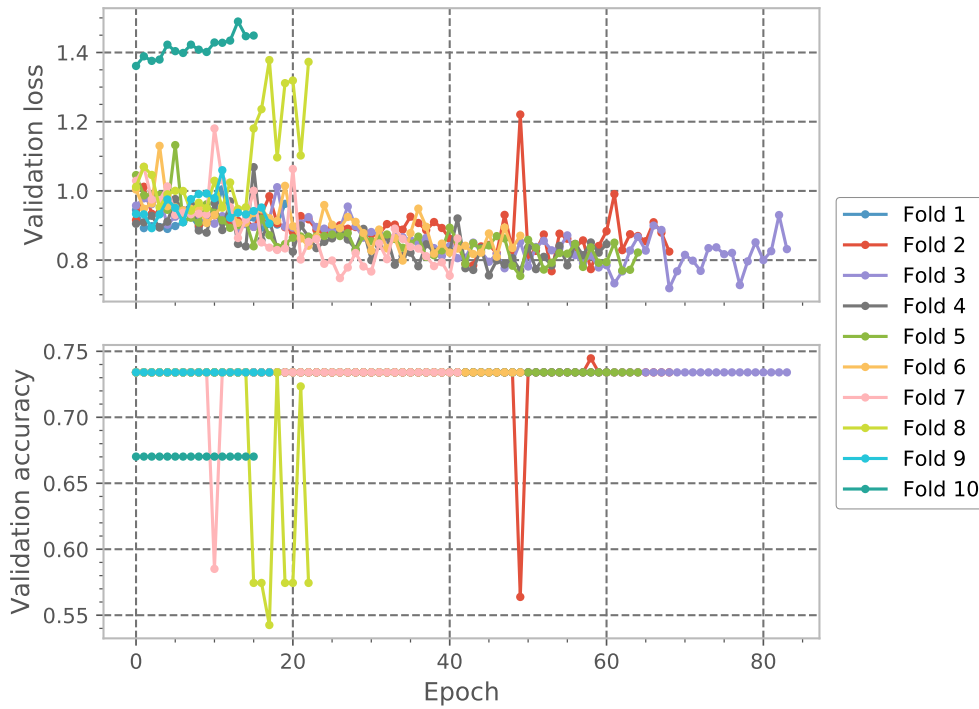


Figure 4.20: Validation history of optimised LSTM-result in the energy use case.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.582	0.399	0.410	0.582	0.466	-0.012
std	0.200	0.199	0.190	0.200	0.211	0.026
min	0.269	0.072	0.155	0.269	0.114	-0.078
max	0.845	0.713	0.731	0.845	0.774	0.000

Table 4.23: Overview of weighted average metrics across all folds for optimised LSTM result.

deviations across all metrics are relatively higher, which is also reflected in the

validation history as well.

Summary of results for LSTM

For the LSTM experiments, two initial experiments and one optimised experiment were carried out. In the first initial experiment, the concern was mainly trying initial hyperparameter settings, based on the data analysis. Further on, in the second initial experiment, we tried improving the model convergence with a different optimiser, increased momentum rate and Nesterov momentum. While the model loss converged well, there was no improvement throughout training and accuracy and loss became saturated. With a weighted average of 0.61 and MCC of 0, the classification task can still be improved.

Moreover, one last experiment was performed, with optimised configurations and early stopping applied. Although one result was presented, many of the tried initial experiments are also found in **Appendix B.3** and gave similar results. While the MCC was negative, the experience overall indicated a certain difficulty when configuring hyperparameters for the LSTM, because the outcome was approximately the same across multiple configurations.

4.2.3 Results for CNN

I_1 initial result

Type	Hyperparameter	Value
Data	Batch Size	1
	Epoch	300
	Resampling Average	D
	Sequence Window	30
Network	Dilation Rate	3
	Dropout	0.3
	Kernel Size	8
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.3
	Nesterov Momentum	False
	Optimiser	sgd
	Padding	causal

Table 4.24: Most important hyperparameter configurations for the initial result.

While previous experiments in the depression use case showed that dilated convolutions are effectively applicable, we further aim to understand its effect in this use case. Additionally, the idea of on-line training is explored, where the training process

progresses one sample at the time. First, a dilation rate of three and a kernel size of eight is used, resulting in a convolution dimension of 24.

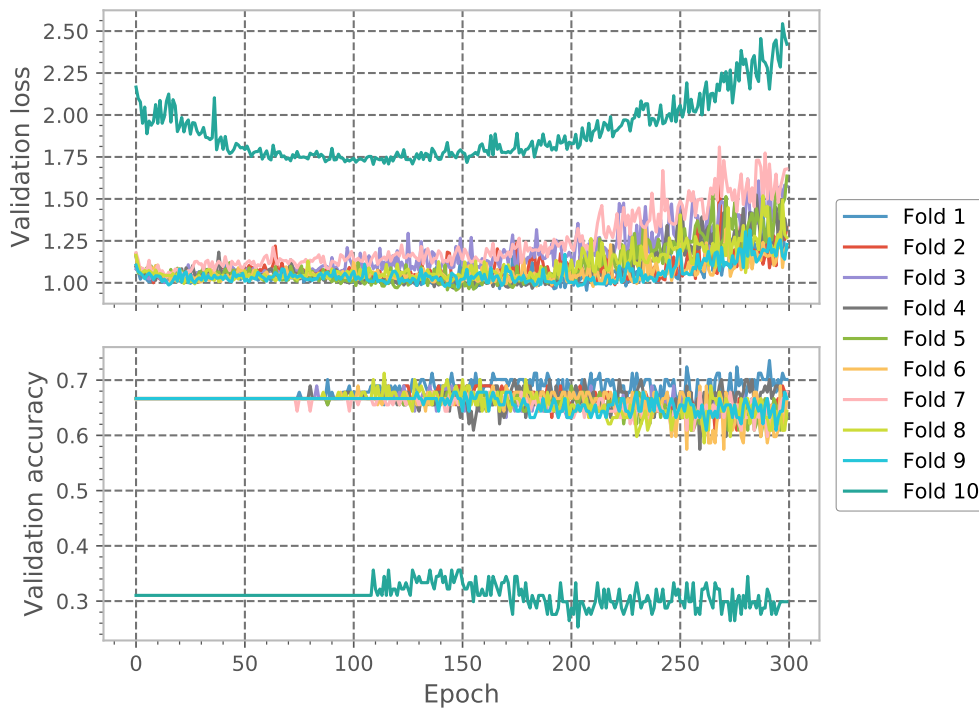


Figure 4.21: Validation history of initial CNN-result in the energy use case.

The training history in **Figure 4.21** shows that the optimisation is consistent across all folds. Additionally, similar to the results for the LSTM-model, the convergence of the accuracy rate is almost constant. Throughout the training, it remains approximately the same, although with minor deviations in the end. Recall from earlier with the LSTM-experiments, that a similar pattern occurred, which can be explained by how the underlying data distribution affects model performance. One explanation is the difficulty of finding patterns, which contributes to randomness in predictions. Too little data or increased noise in the data may contribute to this in various ways. From the metrics shown in **Table 4.25** this is further evident, as the MCC is close to zero.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.560	0.507	0.545	0.560	0.518	0.110
std	0.167	0.164	0.171	0.167	0.164	0.079
min	0.320	0.217	0.265	0.320	0.257	-0.003
max	0.844	0.799	0.792	0.844	0.811	0.257

Table 4.25: Overview of weighted average metrics across all folds for initial CNN result.

Further on, we emphasise that during the training of the CNN, the overall experience of trying different hyperparameters became a less tedious task. Many of the initial experiments showed varied results which further enabled the possibility of trying multiple configurations⁶. The classification task itself, however, can be improved with

⁶Validation history for additional CNN experiments not presented here are found in **Appendix B.4**

further optimisation.

R_1 and R_2 : optimised results

We now look at two results with optimal outcomes, R_1 and R_2 , both using the same hyperparameter configurations. From the previous initial experiment, the kernel size is reduced to five from eight, reducing the temporal dimension to 15. In the first experiment, the model trains for a shorter number of epochs and on most occasions, the training stops around 80 to 90 epochs as seen in **Figure 4.22a**. In the second experiment, however, the model runs around 80 epochs or more.

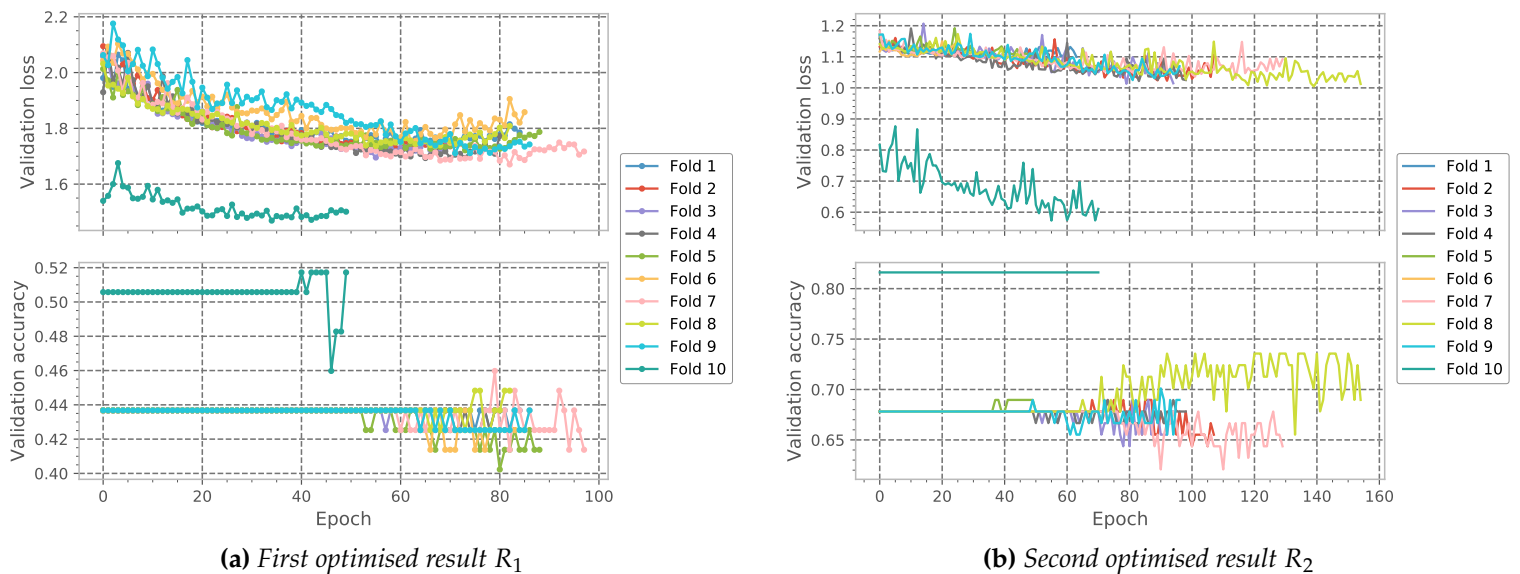


Figure 4.22: Validation history of both optimised CNN-results in the energy use case.

The weighted average of the cross-validation can be seen in **Table 4.26**. Comparing both runs show minor differences, but for the second experiment, which used more time to train, the standard deviation is overall lower, or relatively marginal. On average, the classification rate is marginally better as well. Nevertheless, apart from these minor deviations, the takeaway from the optimised results indicates that the CNN-model adapts better to the data distribution.

Summary of results for CNN

Table 4.26 gives a summary of both optimised experiments. In the first and only initial result, the application of dilated convolutions in combination with on-line training was explored. While the model showed tendencies of overfitting, the progress during the first 150 to 200 training iterations was promising.

Further on, two additional experiments were carried out with equal hyperparameter settings. The kernel size was reduced and the dilation rate was kept from the initial result. In both experiments, the loss convergence was optimal, albeit the accuracy was almost constant throughout the training.

Result Metric (sd)	R_1	R_2
ACC	0.600 (0.141)	0.601 (0.133)
PREC	0.428 (0.191)	0.449 (0.135)
SPEC	0.425 (0.149)	0.426 (0.157)
REC	0.600 (0.141)	0.601 (0.133)
F1	0.484 (0.178)	0.489 (0.143)
MCC	0.039 (0.039)	0.038 (0.048)

Table 4.26: Summary of weighted average evaluations for all optimised CNN-results. Standard deviations are shown in parantheses.

There is a potential for improvement on the classification task itself, considering the accuracy of 0.6 and MCC closer to zero. More importantly, however, the results further substantiate how dilated convolutions can be used efficiently in time-series classification domains.

4.2.4 A comparison on LSTM and CNN

On model training time and predictive performance

Figure 4.23 shows the total training time in seconds for all experiments. Comparably, the time used for both models is approximately the same, although the importance of on-line training with a batch size of one, contributes to an increase in total time used for the CNN-model. As discussed in the previous use case with depression classification, training time only explains the model training complexity to a minor extent. However, as seen in **Figure 4.24**, when comparing this with the predictive performance of the models, the CNN-model shows slightly better classification measures. Additionally, despite being closer to zero, the average MCC is higher for the CNN as well.

With minor deviations in the predictive performance for both models, however, the overall classification task can still be improved. In most cases, both models have an accuracy of around 60%, while the balanced measures like MCC and F1 are closer to zero and 0.5, respectively. One explanation for this can be the hyperparameter settings.

However, another explanation which is potentially more explainable is the size, samples and features of the EV dataset. First, the total observations are closer to 28 000 data points, distributed on nine different EVs. When aggregated to daily measures, the total number of input sequences are approximately 1000, depending on the window size. In the context of machine learning, this is a small dataset, and the classification task for both models can be improved drastically. One possibility includes generating synthetic data.

Another explanation is related to the qualitative features in the dataset. Current features in the EV dataset are not sufficiently descriptive and can affect the model performance. For instance, the battery capacity is only the relative percentage level

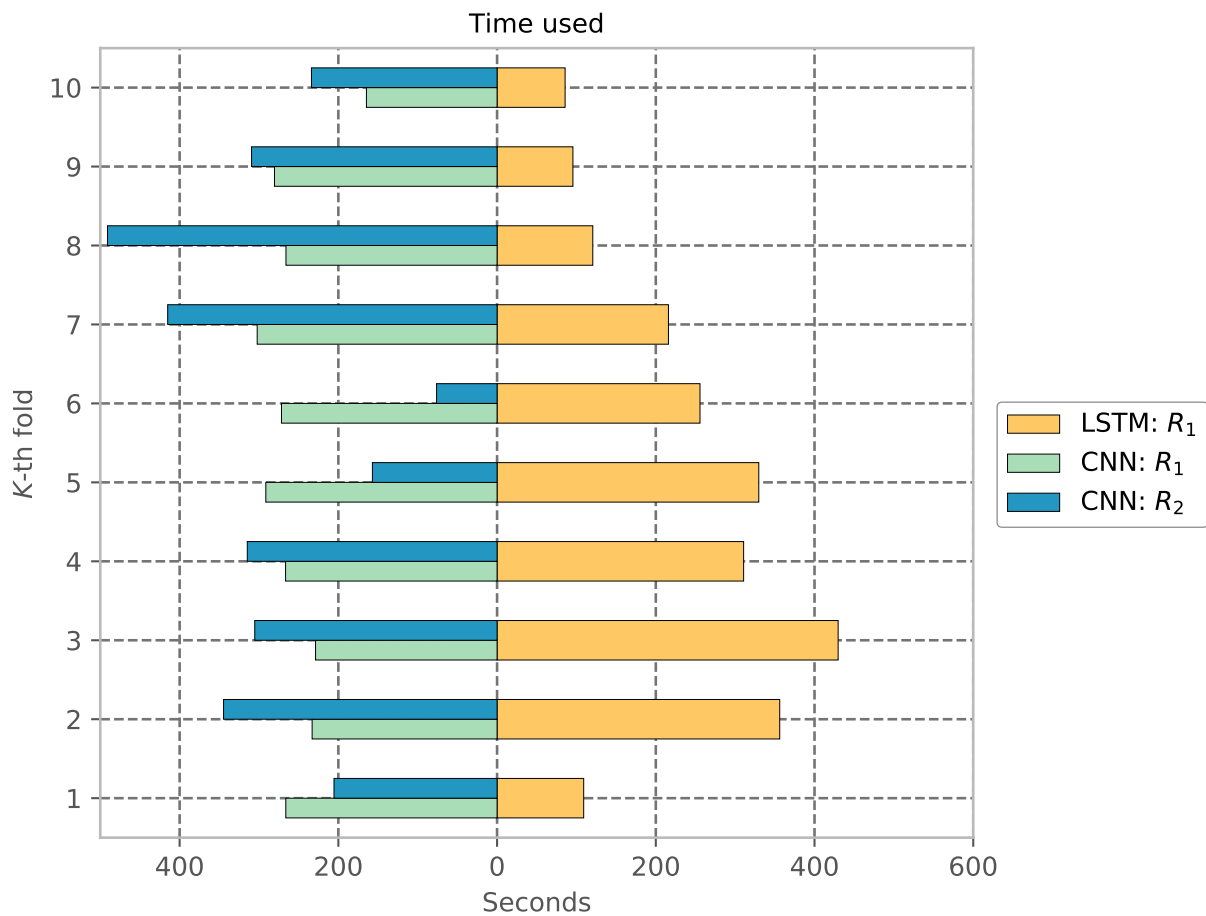


Figure 4.23: Total training time in seconds for each of the models across all results.

for each EV. While different EV types and models have different battery capacities, the percentage is not a qualitative input feature that describes the used capacity well. Possibilities to overcome this problem include collecting additional qualitative features such as kWh or range capacity. Although the results look promising for both models, more qualitative data could potentially give a better basis in the comparison of the CNN and LSTM.

4.2.5 Summary of experiments

For the EV energy prediction use case, six experiments were performed. Overall, both models perform the classification task differently, although there is room for improvement and the differences between both models are minor. In total two initial experiments resulted in one optimised result for the LSTM. On the other hand, one initial result was used to perform two optimised experiments for the CNN.

For the LSTM, the initial experiments were mostly concerned with understanding the effect of the first hyperparameters settings and improving model convergence. The following optimised result aimed at improving the classification rate itself. Different hyperparameters were tried to overcome the problem of constant model accuracy not

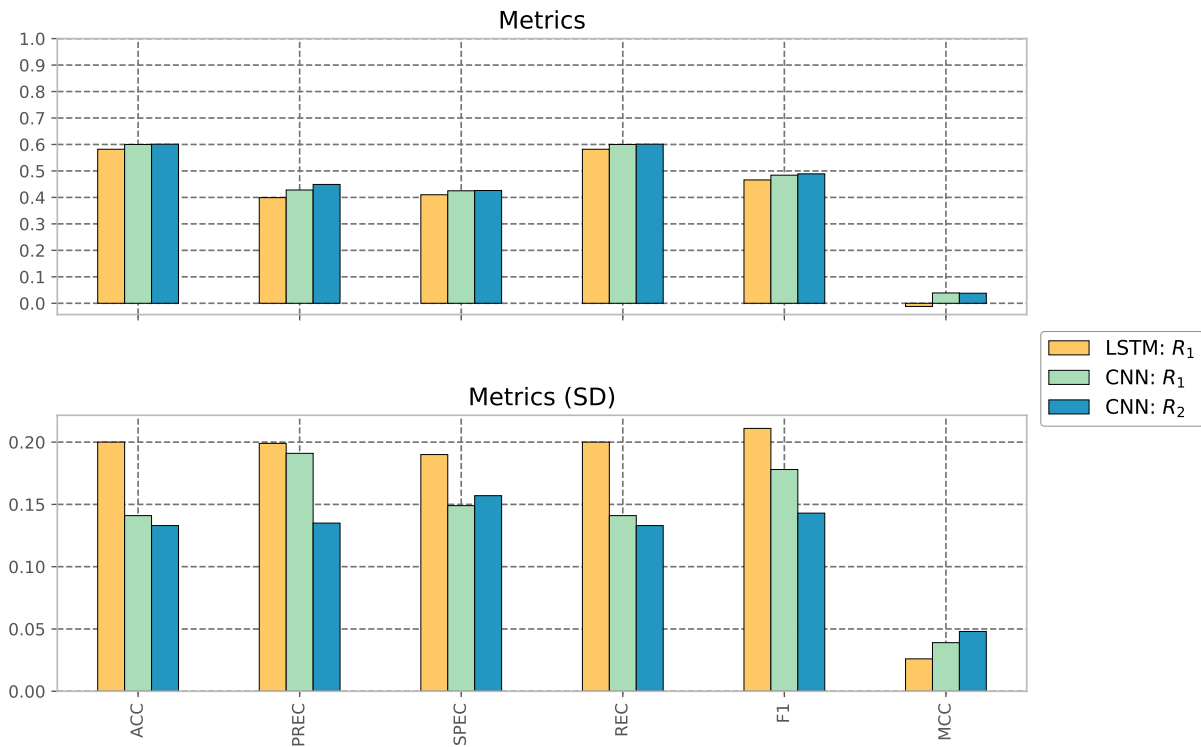


Figure 4.24: The figure shows a summary overview of the weighted average for all results presented earlier, across all models. The top figure shows the metrics whereas the bottom shows the standard deviations.

improving. Some include different optimiser, momentum rate, Nesterov momentum and batch size.

The same pattern emerged for the CNN, although the experiments were more focused on observing the effect of dilated convolutions, which worked seemingly efficient. Comparing the results against the LSTM, the CNN is almost as good. However, in order to have a substantial baseline for this use case, the EV dataset should include more observations or additional qualitative features. In a sense, this would provide better empirical evidence as well, similar to the depression use case where dilated convolutions and training time were more efficient for the CNN.

4.3 U_3 : Football readiness classification

4.3.1 Experiment overview

Problem formulation and experiment preparation

In their paper, Wiik et al. [94] discuss how LSTM is applied for peak detection to determine readiness-to-train of football players. They use a quantitative approach, where the output is continuous. Further on, the peak detection is determined based on thresholded values. Positive peaks are defined based on readiness values over

eight, whereas negative peaks are defined as readiness values below three.

However, in terms of a classification perspective, the problem formulation is almost equivalent. Readiness values have to be thresholded to determine their class representation. One approach is a binary classification formulation, where readiness values are thresholded to a particular score. In the paper, they determine peaks this way, e.g. by thresholding the score to eight. However, from a practical perspective in a classification model, such information would be less informative, because of a crisp output. If readiness is thresholded to eight, a score of seven is classified as "not ready", which is useful information when determining readiness. Moreover, the data analysis earlier suggested that the centre of the distribution of readiness scores reside in the range of five to eight for most players. Hypothetically, a model will thus in most cases classify a player as "not ready" because the average scores are closer to the mid-range.

Nevertheless, to overcome this problem, readiness is in this task defined as the presence of particular variables that determine readiness. Based on the correlation analysis earlier, the problem is formulated as a *multilabel classification problem*, where the chosen variables are mood, stress, soreness and fatigue⁷.

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix} = \begin{bmatrix} \text{Mood} \\ \text{Stress} \\ \text{Soreness} \\ \text{Fatigue} \end{bmatrix} \quad \text{where} \quad \hat{y}_i = \begin{cases} 0 & \text{if } \hat{y}_i < 0.5 \\ 1 & \text{if } \hat{y}_i \geq 0.5 \end{cases}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} \text{Mood} \\ \text{Stress} \\ \text{Soreness} \\ \text{Fatigue} \end{bmatrix} \quad \text{where} \quad y_i = \begin{cases} 0 & \text{if } y_i < 3 \\ 1 & \text{if } y_i \geq 3 \end{cases} \quad (4.1)$$

Equation 4.1: A formalisation of multilabel classification for the readiness classification use case, where the predicted \hat{y} is a distribution (sigmoid output) of four variables, namely mood, stress, soreness and fatigue. The sigmoid output represents probabilities, where a correct classification is determined based on a threshold of 0.5 by default. The true output distribution y is represented as a one-hot encoded distribution, where the encoding is based on a threshold score of 3 by default.

Recall from the methodology chapter that for this use case, the sigmoid activation function is applied in the last layer. The predicted output distribution \hat{y} is continuous, implying a set of probabilities. The classification task is formalised by thresholding the predicted distribution. The outputs are thresholded to 0.5 to determine the presence of the i -th class \hat{y}_i , as shown in **Equation 4.1**. Furthermore, the chosen input features are readiness, stress, mood, soreness and fatigue, which is based on the correlation analysis earlier. The variables are scaled between zero and one because of different scales on readiness scores compared to the other variables.

⁷A multilabel classification problem is where an instance or a sample belongs to multiple classes, compared to a multiclass classification problem where the sample may only belong to one class. For instance, given a player report, readiness can be determined by the presence of stress, mood, fatigue (multilabel) or just the readiness score (multiclass).

Overview of narrative

For this use case, multiple approaches were tried before deciding a multilabel formulation. Most efforts did not give any reasonable results. First, experiments with multiclass classification were carried out. Hence, the output was defined as a readiness score from one to ten and represented as a one-hot encoded vector. As hypothesised in the above section, most predictions resided in the range five to seven.

Secondly, another formulation was concerned with categorising the readiness scores into three classes, namely, *not ready*, *uncertain* and *ready*. The classes were determined based on threshold ranges where each class belonged to the range one to three, four to seven and eight to ten, respectively. In a way, this is simply a smoothing of the data distribution. Similar to the first case, most predictions were classified into the *uncertain*-class.

Both the abovementioned methods are relatively alike and use a thresholding approach. This method is similar to the one discussed in the quantitative approach discussed by Wiik et al. [94]. Equivalently formulating the classification task showed no improvement on this baseline either.

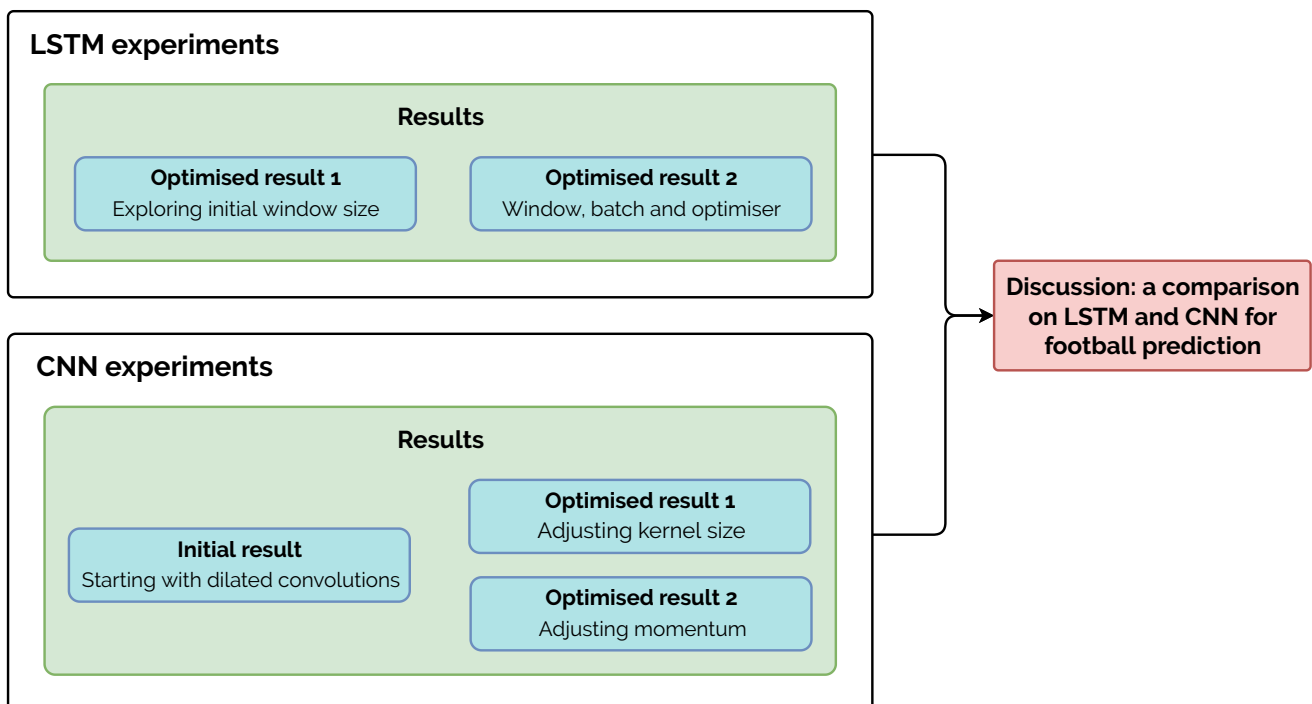


Figure 4.25: Experiment overview for football readiness classification use case. We perform two experiments with LSTM and three experiments with CNN.

Hence, to understand how readiness can be determined in an alternative way and be beneficial to coaches from a system perspective, a multilabel classification formulation was chosen. The presented results showcase the potential application area for football readiness prediction. As mentioned, multiple efforts were tried on different formulations of the classification problem. Additionally, some configurations highlighted in **Appendix B.5** were also tried out. However, because of time limitations,

in particular, the chosen results only reflect a subset of the experiments. Each is concerned with discussing the various configurations used. The overview of these experiments is seen in **Figure 4.25**.

4.3.2 Results for LSTM

This section presents three different results for LSTM, namely R_1 , R_2 and R_3 . Recall that for this use case, as described in the methodology, we present the optimised results because many of the initial results and configurations resulted in approximately the same outcome or did not show much improvement.

R_1 : first result

Type	Hyperparameter	Value
Data	Batch Size	15
	Epoch	300
	Resampling Average	D
	Sequence Window	30
Network	Dropout	0.3
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.4
	Nesterov Momentum	False
	Optimiser	rmsprop

Table 4.27: Most important hyperparameter configurations for the first optimised LSTM-result.

The first experiment is concerned with trying the initial configurations. First, a relatively larger batch size of 15 is used, compared to Wiik et al. [94] in which they report a batch size of four for the presented results. Additionally, the input sequence window size is 30 data points, which is equivalent to a monthly timespan. Moreover, the initial optimiser is RMSprop, similar to what is reported by Wiik et al. [94]. The batch size is arbitrarily set, whereas the sequence window starts with 30, as it is determined a reasonable timespan to capture regularities in readiness.

For the above-mentioned configuration, the cross-validation averages are seen in **Table 4.33**. For a multilabel classification problem, we evaluate the model based on the number of correctly predicted labels in the output distribution. The accuracy is 0.899, implying a good predicting of the overall distribution. Moreover, the MCC of 0.283 is positive, which indicates a positive correlation in the predictions and observations. However, with the value being closer to zero with overall higher deviations, there is room for improvement. Additionally, despite being somewhat lower, the precision, recall and F1-score are comparable to the presented result by Wiik et al. [94].

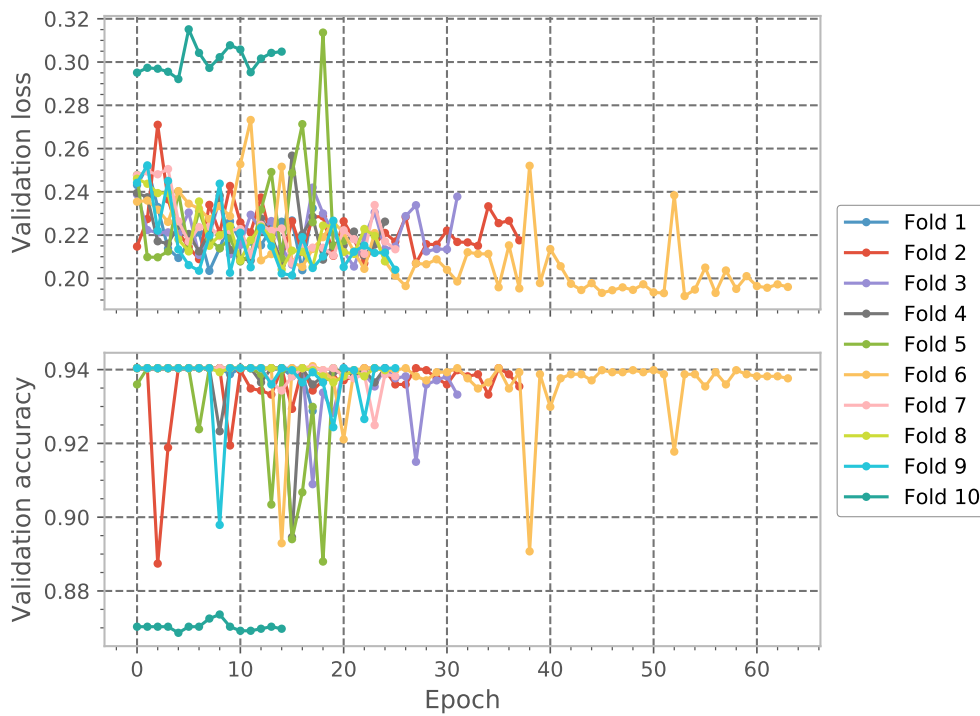


Figure 4.26: Validation history of first optimised LSTM result in the football use case.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.899	0.915	0.152	0.985	0.947	0.283
std	0.028	0.020	0.059	0.017	0.016	0.218
min	0.870	0.890	0.072	0.944	0.924	0.111
max	0.956	0.959	0.260	1.000	0.978	0.701

Table 4.28: Overview of weighted average metrics across all folds for first optimised LSTM result.

The overall training progression illustrated in **Figure 4.26** explains some variations during training. Although the model convergence in most cases looks promising, the training stops approximately after 20 to 30 iterations. The validation accuracy is almost constant throughout training, whereas the loss differs between validation folds. For instance, for fold five, it looks like the LSTM overfits. On the other hand, for the sixth fold, the training goes on for longer epochs and converges better. The confusion matrix for this evaluation is seen in **Figure 4.27**.

The confusion matrix explains the presence of a variable in the output distribution. Based on the problem formulation earlier, the threshold score is three, in which *Present* indicates whether the outcome variable is present or not. The presence of a variable implies a rating higher than three. From **Figure 4.27**, it is seen a confusion in the prediction of soreness. In most cases, the distribution of scores in mood, stress and fatigue looks to be uneven. We derive this from the proportion between false negatives (top right) and true negatives (bottom right) for the mentioned variables. However, for soreness, there are somewhat larger confusions. Most probably, this

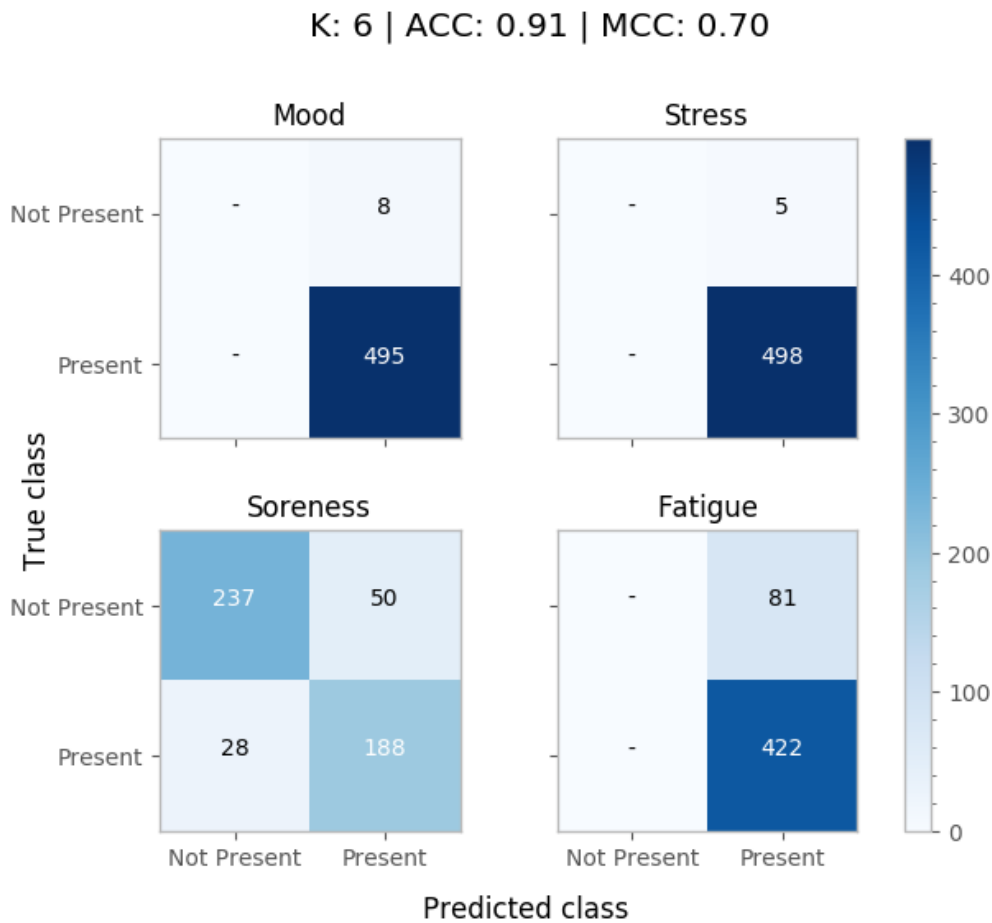


Figure 4.27: Confusion matrix for the best performing model in the first optimised LSTM result.

may be related to higher variations in soreness scores, and thus, imbalances in the dataset leading to such classifications.

Nevertheless, the convergence of the model is desirable for this particular validation. An MCC of 0.70 indicates a stronger relationship between predictions and observations. Lastly, an accuracy of 0.91, further implies a stronger classification rate, which showcases a classification model with good improvement potential as well.

R_2 : second result

The second result presents an alternative experiment, where the batch size, sequence window and activation function are tuned. Compared to the previous configuration, the batch size is reduced from 15 to 10, whereas the sequence window increase from 30 to 40. Moreover, the activation function changes from ReLU to tanh. Most importantly, however, we performed multiple experiments with different optimisers, sequence window lengths and batch sizes. The most promising outcomes were achieved with configurations close to these. The current best evaluations with the abovementioned settings are shown in **Table 4.34**.

Type	Hyperparameter	Value
Data	Batch Size	10
	Epoch	300
	Resampling Average	D
	Sequence Window	40
Network	Dropout	0.3
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.001
	Momentum Rate	0.4
	Nesterov Momentum	False
	Optimiser	rmsprop

Table 4.29: Most important hyperparameter configurations for the second optimised LSTM result.

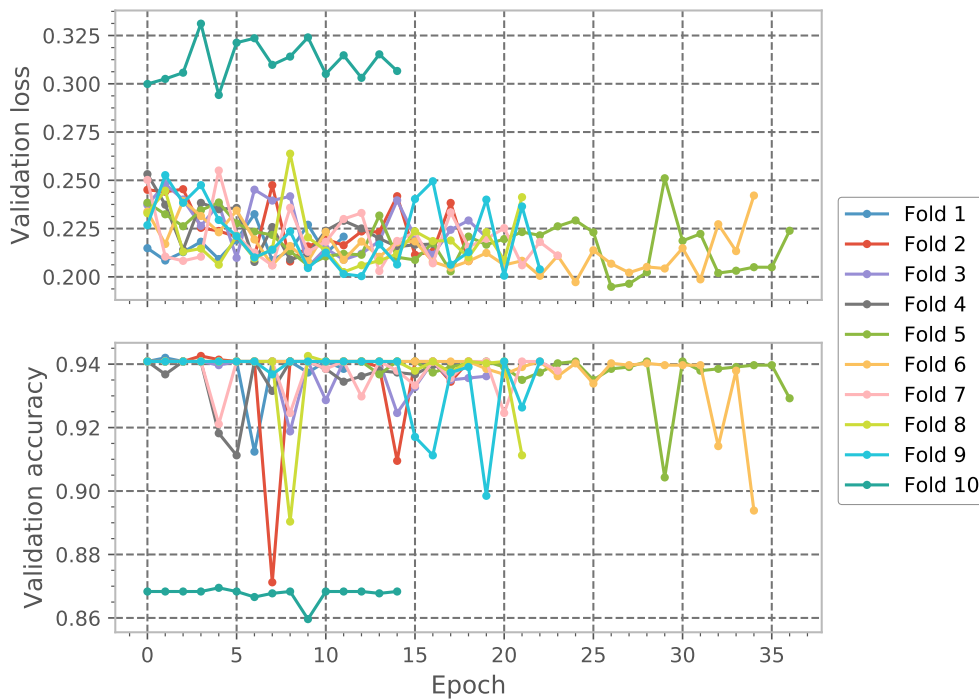


Figure 4.28: Validation history of second optimised LSTM-result in the football use case.

Overall, the results are relatively similar to what is achieved in the previous experiment. Across all evaluations, there are marginal differences. Although the deviations are somewhat higher for the MCC in the first experiment, the results are arguably quite similar. One reason for this may be related to the dataset itself.

The loss landscape of the underlying distribution may be less noisy, in the sense that most players recorded in the dataset have consistent score reports over time. In other words, the target labels will have an underlying imbalance in the dataset. Classifying the presence of these variables may thus not sufficiently explain the various conditions of a player. For instance, the model may be good at predicting low

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.900	0.915	0.155	0.986	0.948	0.277
std	0.030	0.022	0.057	0.014	0.017	0.203
min	0.858	0.888	0.075	0.952	0.918	0.052
max	0.957	0.959	0.258	1.000	0.979	0.708

Table 4.30: Overview of weighted average metrics across all folds for second optimised LSTM result.

stress, good mood, soreness and fatigue. However, when there are fewer instances of lower scores, like high stress and bad mood, etc., the model may not capture such cases well. The confusion matrix presented in the first experiment showed a similar pattern, in which the same is observed for this experiment, as illustrated in **Figure 4.29**, which shows the confusion matrix for the best performing model. Additionally, this problem is also highlighted by the specificity evaluation, which measures the true negative rate, and is close to zero, compared to the other metrics. Conclusively, the model classifies positive scores (above 3) efficiently but is inefficient in predicting negative scores (below 3).

K: 6 | ACC: 0.91 | MCC: 0.71

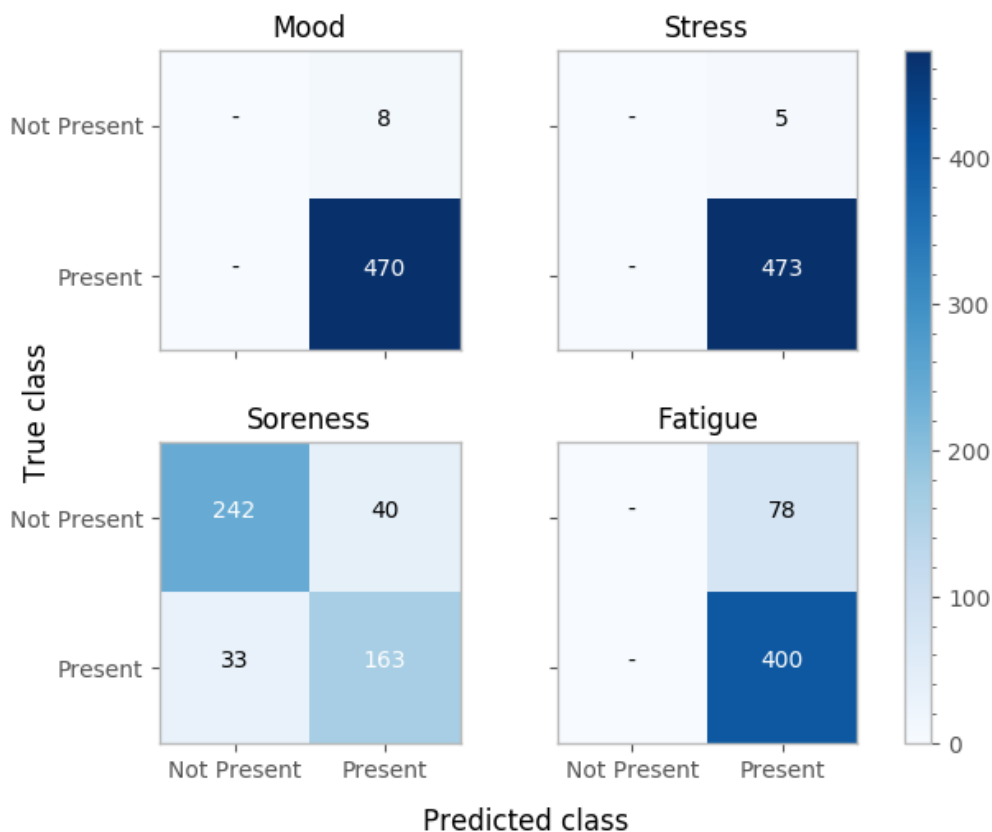


Figure 4.29: Confusion matrix for the best performing model in the second optimised LSTM result.

Summary of results for LSTM

Result Metric (sd)	R_1	R_2
ACC	0.899 (0.028)	0.900 (0.030)
PREC	0.915 (0.020)	0.915 (0.022)
SPEC	0.152 (0.059)	0.155 (0.057)
REC	0.985 (0.017)	0.986 (0.014)
F1	0.947 (0.016)	0.948 (0.017)
MCC	0.283 (0.218)	0.277 (0.203)

Table 4.31: Summary of weighted average evaluations for all optimised LSTM results. Standard deviations are shown in parantheses.

Table 4.31 gives a comparison overview of both results presented for the LSTM experiments. While the experiments are not directly comparable to what is reported by Wiik et al. [94], the presented results showcase the application of multilabel classification for this use case.

Moreover, the overall experience throughout hyperparameter tuning is varied. In many cases, the LSTM is very consistent in the classification task, regardless of configurations. However, some configurations and ranges work better than others, although the improvement in evaluations is marginal. Nevertheless, the most important takeaway is the ability of the model to predict negative classes efficiently. The LSTM classifies positive scores well. However, negative scores (scores below 3) are wrongly classified in almost all cases. One approach to overcome this is to tune hyperparameters more extensively. However, the preferable approach is to include additional variations in the dataset, which include more instances of lower scores.

4.3.3 Results for CNN

This section presents the results for the CNN, where one initial result is discussed, followed by three results with different hyperparameter settings. They are all discussed on a general level to further showcase the potential application of CNN. We emphasise that additional experiments should be carried out with different hyperparameters than those presented here.

I_1 : initial result

Similar to the LSTM experiments, the initial experiment uses a batch size of 15 and a sequence window of 30. Moreover, CNN-specific hyperparameters like dilation rate and kernel size are three and four, respectively. Moreover, the momentum rate is 0.4, whereas the learning rate is 0.005, with SGD being the optimiser.

Figure 4.30 shows the overall training history for these configurations. Initially, there are two notable observations. First, the model converges well the first 25 epochs, before the validation loss increases and accuracy drops gradually. While this is

Type	Hyperparameter	Value
Data	Batch Size	15
	Epoch	300
	Resampling Average	D
	Sequence Window	30
Network	Dilation Rate	3
	Dropout	0.3
	Kernel Size	4
	Layers and nodes	[150, 75, 50, 25]
	Learning Rate	0.005
	Momentum Rate	0.4
	Nesterov Momentum	False
	Optimiser	sgd
Padding	causal	

Table 4.32: Most important hyperparameter configurations for the initial CNN result.

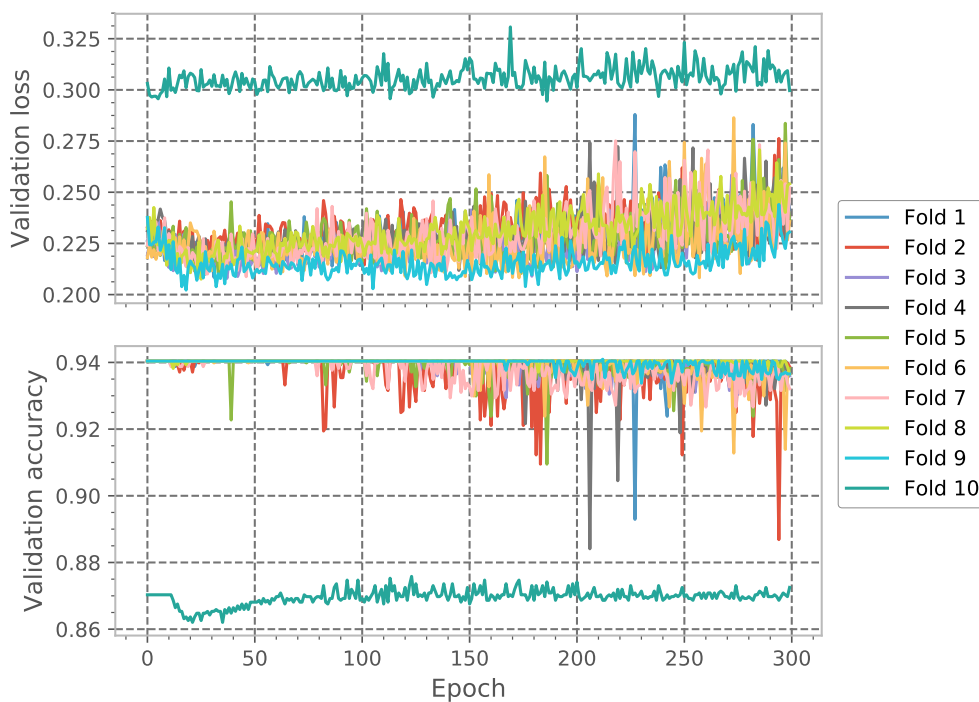


Figure 4.30: Validation history of first initial CNN-result in the football use case.

characteristic for model overfitting, the decrease in accuracy and increase in loss is marginal. It can be argued that the model overfits in this case, although it may be a good indication of robustness, as the loss is not increasing at a higher rate.

Second, for the last validation fold, the training history is unusually different, compared to the average throughout the other folds. There may be multiple reasons for this. Arguably, one explanation relates to the dataset partitioning, which is a pattern we observed earlier as well. Recall from the depression use case, where it

was evident that this problem was because of multimodality in the data distribution.

Overall, while it is difficult to describe which hyperparameters best explain the training progress here precisely, the outcome proves as a reasonable basis for further exploration. The convergence in loss after the first 25 epochs is desirable, in which the accuracy is almost constant during the same training period.

R_1 : first result

We try a different kernel size and increase the length to five, from four in the initial experiment. With a dilation rate of three, similar to earlier, the overall size of the sliding window is 15.

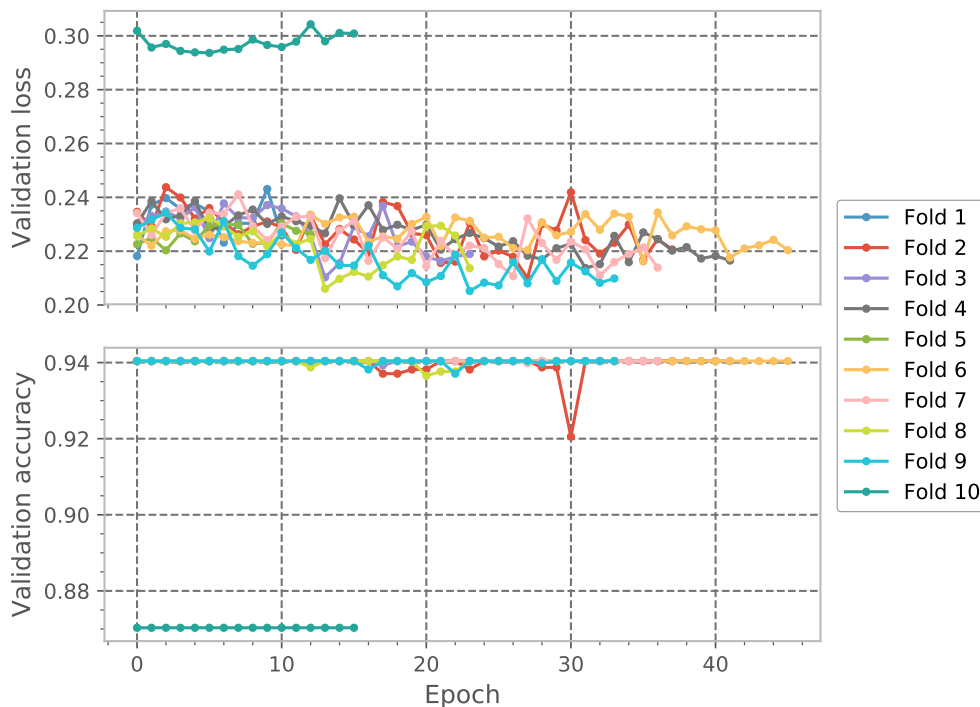


Figure 4.31: Validation history of first optimised CNN-result in the football use case.

The training history shown in **Figure 4.31** shows a slowly converging model. Across most folds, the decrease in validation loss is almost linear, while the accuracy is constant. While the data distribution and partitioning may explain the outlier fold, this is not evident in terms of training convergence. One hypothesis, however, is how saturated gradients or bad optimiser can contribute to slower convergence. Multiple factors can determine this, such as initial weights of the model, the optimiser itself, the learning rate or momentum. To further explore one of them, we look at various configurations on the momentum rate in the second experiment. Not all options are tried, as justified earlier, and some outcomes are found in **Appendix B.5**.

Similar to earlier results and LSTM predictions, there is an evident drop in specificity, the true negative rate. The CNN, similar to the LSTM, shows the inability to capture the imbalances in the dataset efficiently. More specifically, this means that the model

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.892	0.909	0.139	0.993	0.946	0.222
std	0.033	0.024	0.055	0.008	0.015	0.245
min	0.838	0.875	0.072	0.977	0.924	-0.034
max	0.956	0.959	0.253	1.000	0.978	0.666

Table 4.33: Overview of weighted average metrics across all folds for first optimised CNN result.

fails to capture the cases where stress, mood, fatigue and soreness scores are lower than three. As we have discussed, this may be due to score imbalance, although one way to approach the problem is to include additional features that better describe these output labels. On the other hand, the recall and F1-score are 0.993 and 0.946, which shows that the CNN also efficiently classifies instances that have scores higher than three.

R_2 : second result

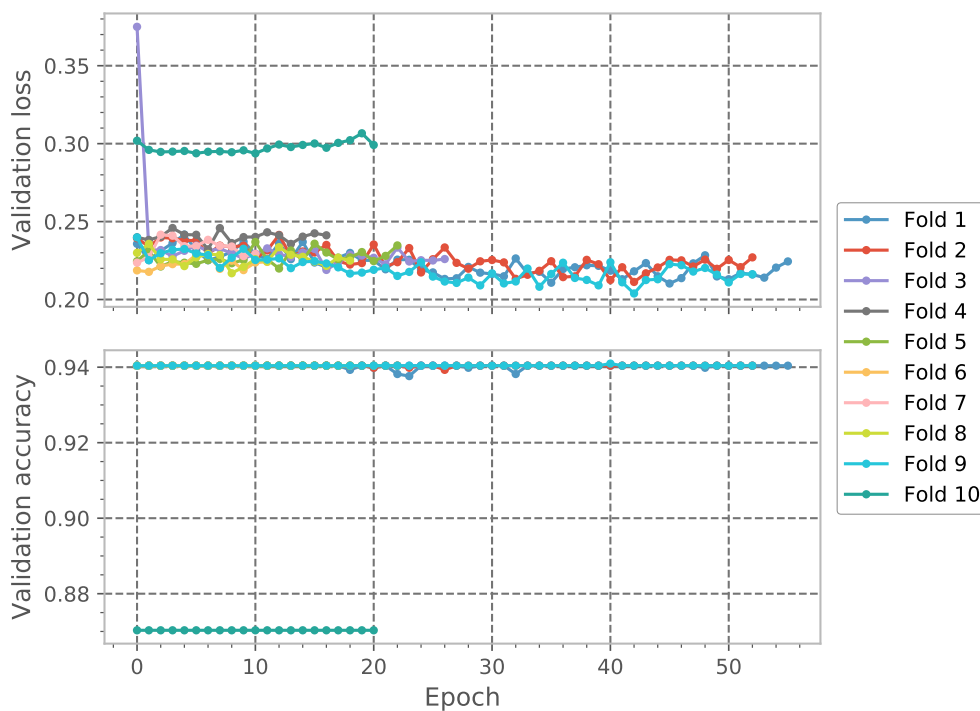


Figure 4.32: Validation history of second optimised CNN-result in the football use case.

In this experiment, the momentum rate is reduced from 0.4 to 0.2. Recall the hypothesis proposed in the previous experiment, where model convergence may be related to multiple factors. Although the loss convergence did decrease, the rate was slower with some variations throughout the training progress. By adjusting the momentum rate, the variations can be reduced, although the decrease in loss becomes potentially slower.

	ACC	PREC	SPEC	REC	F1	MCC
mean	0.887	0.907	0.134	0.994	0.945	0.147
std	0.038	0.024	0.055	0.010	0.016	0.249
min	0.811	0.875	0.072	0.974	0.921	-0.026
max	0.956	0.959	0.247	1.000	0.978	0.585

Table 4.34: Overview of weighted average metrics across all folds for second optimised CNN result.

Figure 4.32 further underpins this hypothesis. The convergence of the validation loss is almost linear, similar to earlier, although the variations are reduced notably. However, it looks like there are marginal differences between both results, which can also be seen in **Table 4.34**. Despite fewer variations, the overall progression is almost the same for both experiments. The validation accuracy remains constant, whereas the only difference in terms of loss convergence is smaller variations across all validation folds.

Summary of results for CNN

Result Metric (sd)	R_1	R_2
ACC	0.892 (0.033)	0.887 (0.038)
PREC	0.909 (0.024)	0.907 (0.024)
SPEC	0.139 (0.055)	0.134 (0.055)
REC	0.993 (0.008)	0.994 (0.010)
F1	0.946 (0.015)	0.945 (0.016)
MCC	0.222 (0.245)	0.147 (0.249)

Table 4.35: Summary of weighted average evaluations for all optimised CNN results. Standard deviations are shown in parantheses.

The summarised results are seen in **Table 4.35**. Similar to the LSTM experiments, the evaluations are consistent across experiments, in which the most notable aspect is related to the drop in specificity. Overall, we experience the CNN to be flexible and faster to train, although finding optimal hyperparameters became a tedious task throughout the experiments. Moreover, the differences across experiments are marginal, similar to what was experienced with the LSTM. Arguably, this implies a certain consistency in the dataset, although the drop in specificity indicates otherwise.

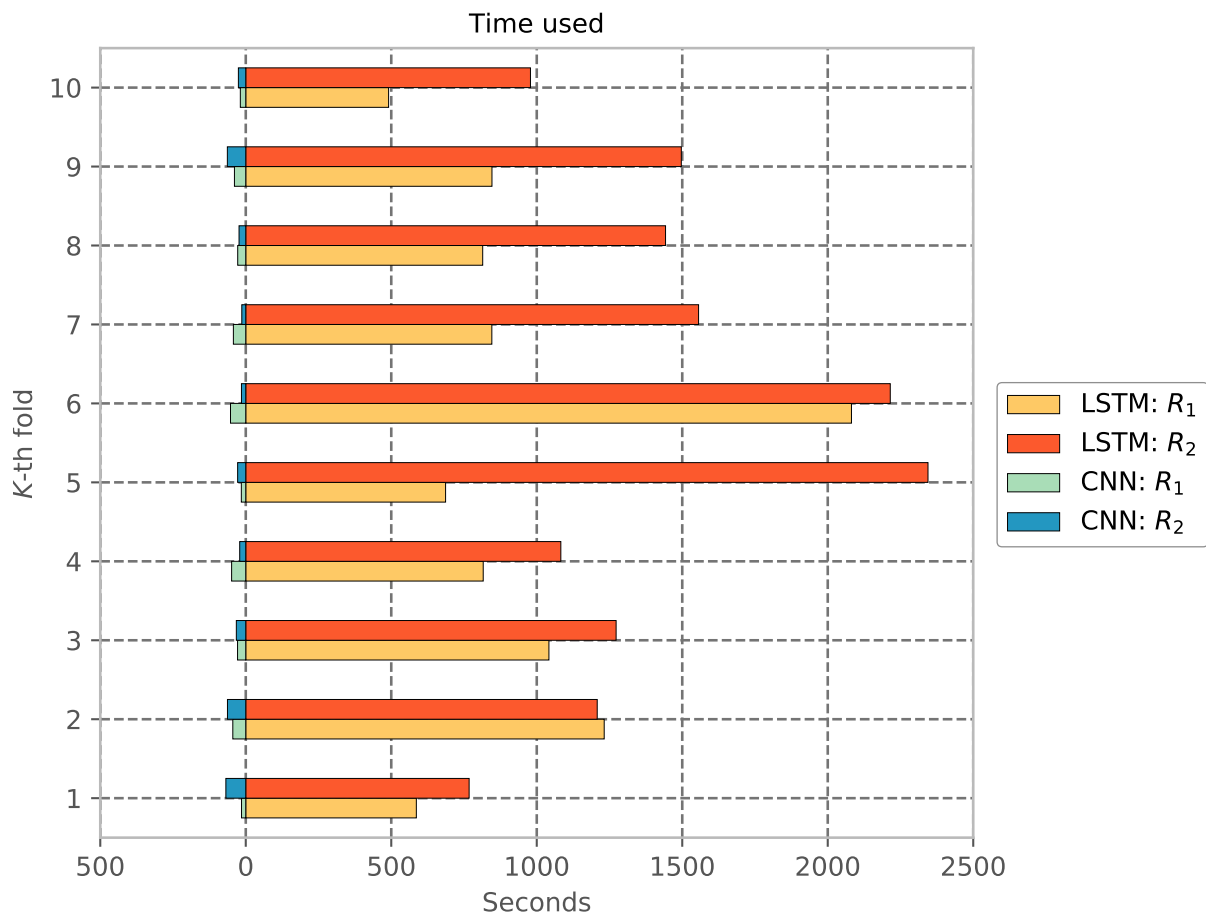


Figure 4.33: Total training time in seconds for each of the models across all results and validation folds.

4.3.4 A comparison on LSTM and CNN

On model training time and convergence

To illustrate the experiences when training the LSTM and CNN, we analyse the training time used in seconds, as shown in **Figure 4.33**. Similar to what we emphasised in earlier use cases, the time used in seconds varies between system, testbed and hyperparameters. However, in most cases, the CNN is approximately 16 times faster, when comparing the training time in seconds, and achieves similar results to the LSTM, if not marginally better.

Comparing the used seconds in context with the number of training iterations, both models use almost the same amount of epochs to learn the data distribution. However, a common factor throughout training is a slower convergence, where the progress was almost constant for both models. There was no observed increase in validation accuracy, and we discussed certain aspects of how to overcome this problem and what the reasons may be.

One general problem is potentially a bad optimiser resulting in slower convergence. To test this hypothesis, we tried adjusting the momentum in our CNN-experiments.

Additional experiments highlighted in **Appendix B.5** also showcase different optimisers that were tried, although many of them did not improve on the best results. Nevertheless, despite lower convergence for both models during training, they both perform equally well in terms of multilabel classification. The LSTM uses more time to train compared to the CNN, in which the latter achieves similar results as well. However, another shortcoming of both models is the ability to capture the class imbalance efficiently.

On predictive performance and class imbalance

As discussed earlier, the results of the CNN are promising when compared to the LSTM, which uses far more time to train and achieves almost the same results. However, the most notable aspect for both models is their inability to predict the negative class efficiently. This is seen in the high drop in specificity across all experiments, as shown in **Figure 4.34**.

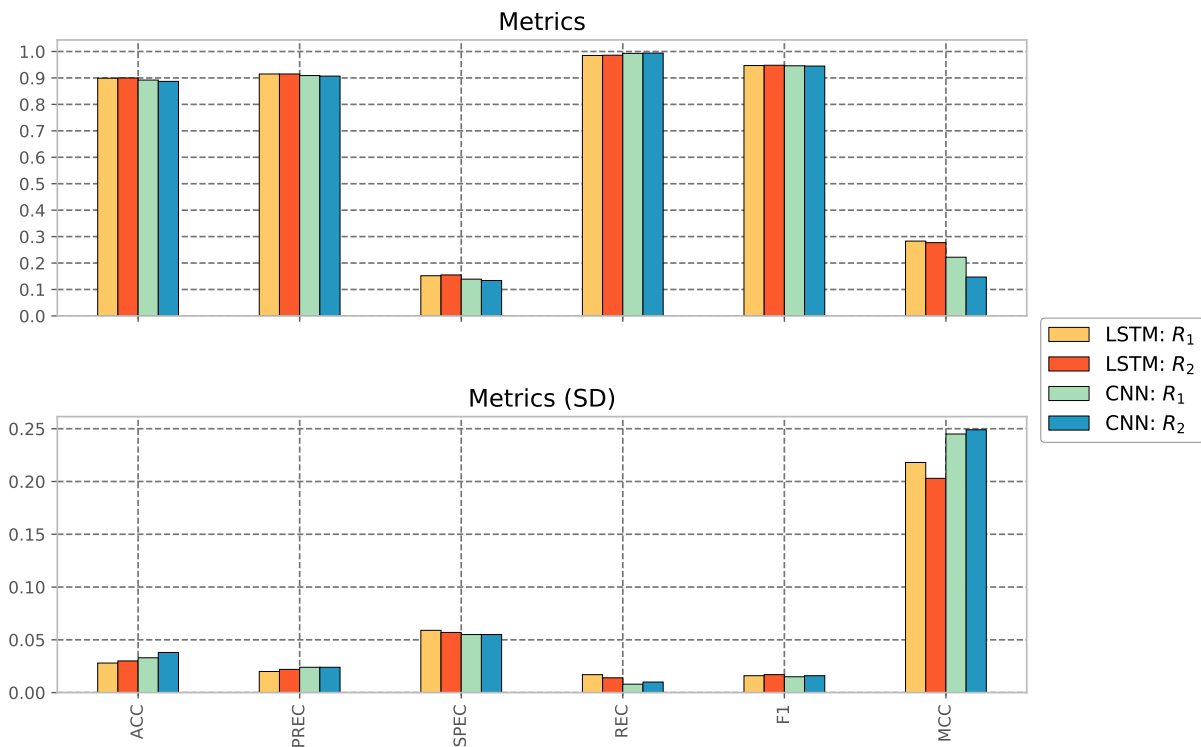


Figure 4.34: A summary overview of the weighted averages across all results which were presented earlier. Top figure shows the metrics for each result, whereas the bottom shows the standard deviation.

Recall from the problem formulation earlier, that the output distribution represents a binary outcome of whether a reported score is less than or larger than a given threshold, which is three by default. A low specificity implies that the negative class for each target variable, defined as scores below three, are not captured efficiently. To further understand which variables the models are most sensitive to, the outcome can be analysed through the generated confusion matrices.

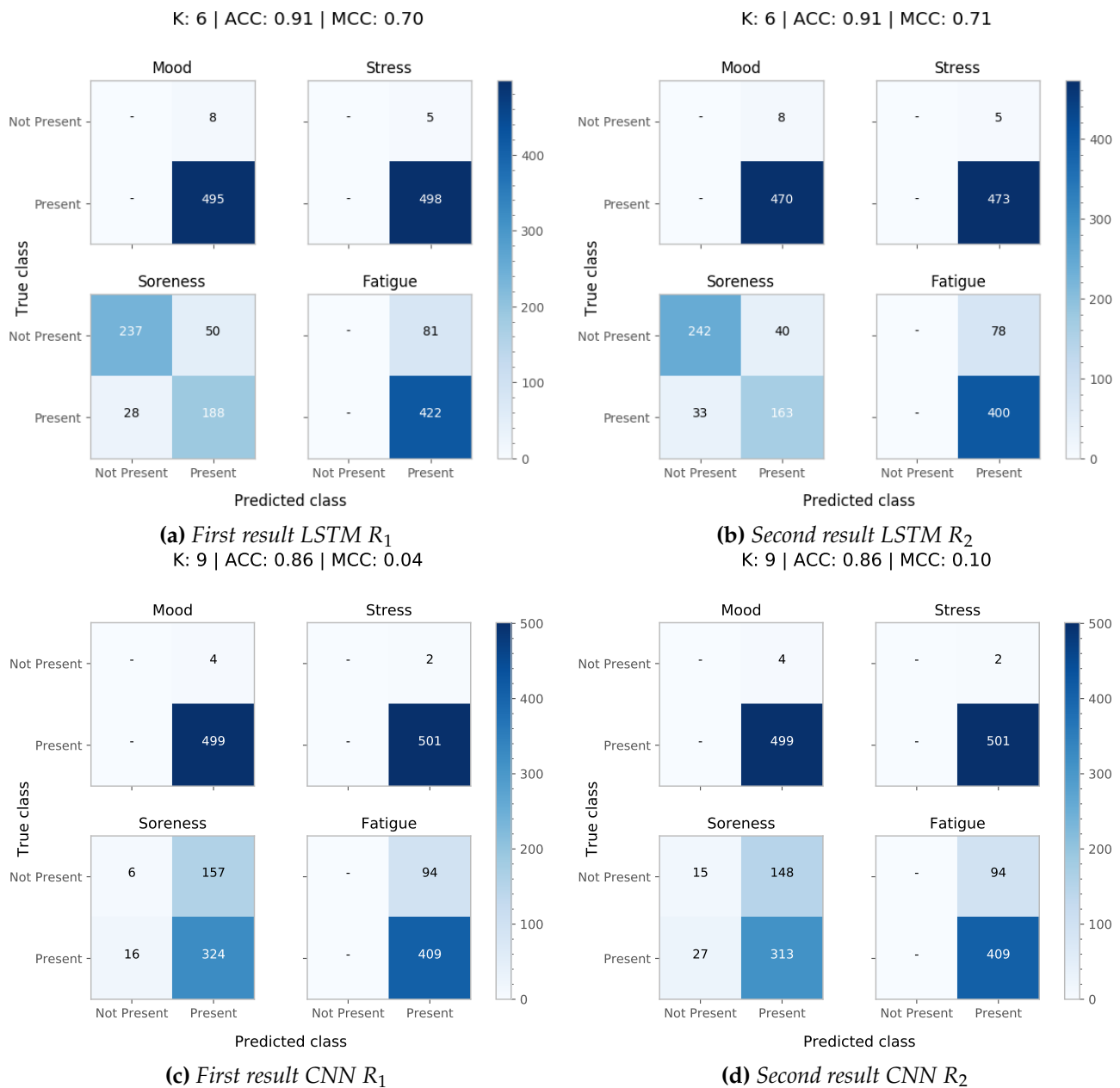


Figure 4.35: Confusion matrices for best performing models across all experiments. For each experiment there are four confusion matrices, each representing a binary confusion matrix for a given variable. The negative class "not present" implies that scores below three are not present for a given variable, whereas the positive class "present" explains the presence of a score higher than three, the default threshold.

From **Figure 4.35**, it is seen that predictions on both the LSTM and CNN classify mood and stress scores effectively. The class *present* for each variable indicates the presence of a score greater than three, whereas the class *not present* indicate the presence of scores less than three. While mood and stress scores in most cases are above three, scores on soreness and fatigue are usually lower than this threshold and are misclassified often.

The number of misclassifications for the best performing models is higher for the CNN than the LSTM. The CNN is more confident on higher scores on soreness and fatigue in cases where the scores are actually lower (more sore and more tired)⁸. On the other hand, the best performing LSTM-model is better at classifying the soreness variable, although the same pattern is observed for the presence of higher fatigue scores, where the actual scores are below three. Apart from the best-performing models, however, the general misclassification patterns are seen for the LSTM-models as well. The best-performing model may thus not be the most representative, but showcases how the models are sensitive to the class imbalance in the multilabel classification context.

4.3.5 Summary of experiments

For the football use case, there are two LSTM-experiments and two CNN-experiments. Notably, each experiment can be explored further through more hyperparameter optimisation. However, without extensive efforts, we show that both models are applicable for time-series classification in the multilabel classification context. Furthermore, the results suggest that the differences in both models are marginal, although the CNN uses far less time on the current testbed. The CNN was easier to train as well, which lowered the threshold for trying more hyperparameter configurations than for the LSTM, despite this becoming a tedious task over time because of minor improvements.

This became a recurring pattern throughout the experiments. Regardless of the model, there were marginal improvements, or close to no improvements at all. The overall convergence in the loss was very slow in which the validation accuracy remained constant throughout the training. Although multiple options were tried out, the best results achieved were discussed. Arguably, one reason is whether this might be related to the optimisation of the network itself. Some configurations were tried out for the CNN, like optimiser and momentum, all highlighted in **Appendix B.5**. However, with no significant improvements, it is difficult to conclude what may be the reason. There is no substantial comparison baseline, and it is emphasised that further experiments can be performed to explore this problem.

4.4 Discussion

This section covers a general level discussion based on the findings for each use case. Although we presented a comparison and discussion on a use case level, this section highlights the main concerns and aspects. The discussion divides into three concerns, which focus on the observations made throughout the experiments. The first concern relates to model training time, whereas the second concern addresses the importance of data and how the models learn the data distribution. The last concern relates to the architectural differences of each model.

⁸The variable meanings are explained in more detail by Wiik et al. [94] in their paper on predicting peak readiness-to-train.

4.4.1 Faster training times

One common factor, experienced throughout the experiments relates to the model training time. Earlier, the model training times were discussed for each use case. Although used time varies between system testbed and configurations used, it explains the complexity and efficiency of the models to some extent. From **Figure 4.36** we see how the CNN uses far less time than the LSTM across all experiments, apart from the EV use case being an exception. Moreover, on an average level, the differences in CNN training times are approximately ten times lower.

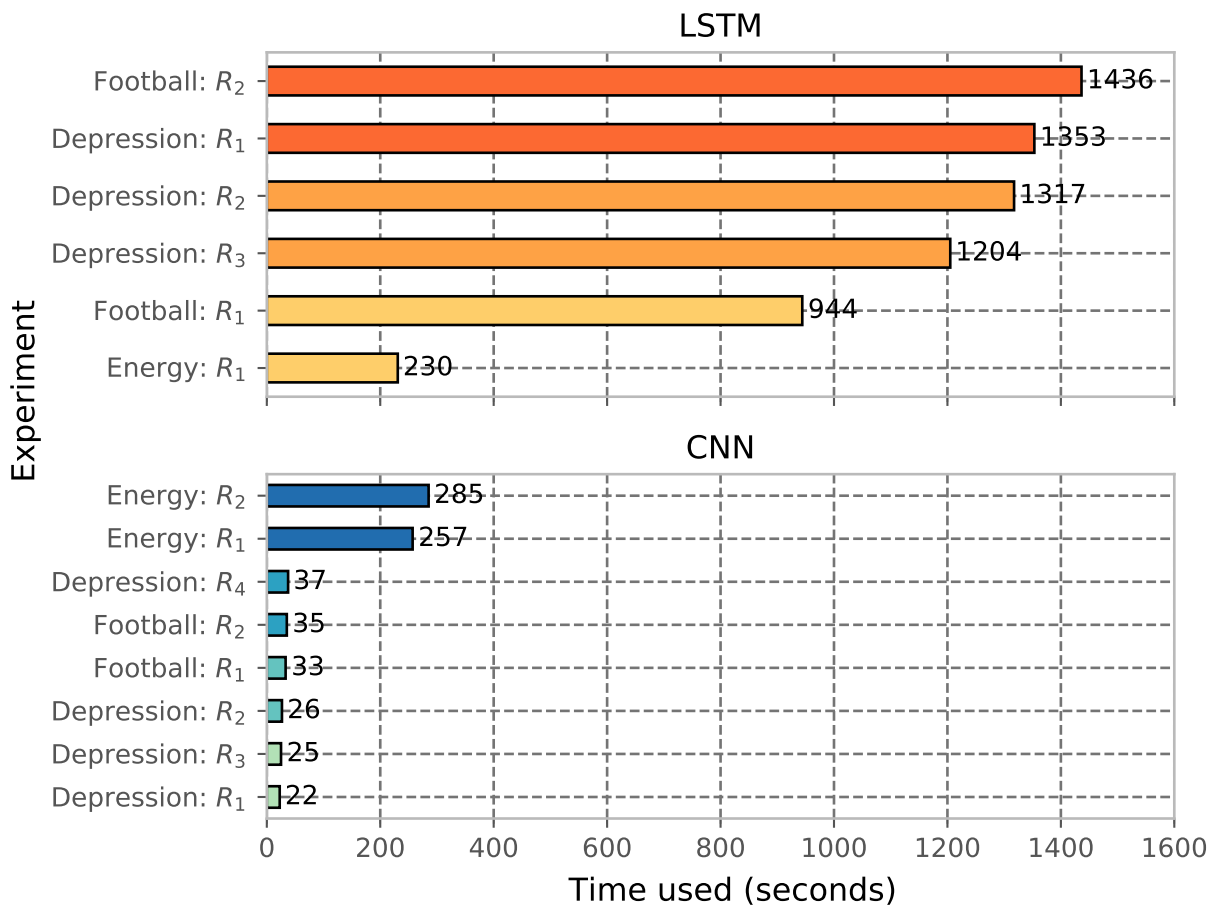


Figure 4.36: The figure shows the average training times across the validation folds for all experiments in this thesis. The x-axis denotes a given experiment whereas the y-axis denotes the total time used in seconds. It can be seen that the CNN in most cases is more than ten times faster to train.

Arguably, the exception of the EV use case can be explained by the dataset size, which is smaller compared to the two other datasets used in this thesis. Additionally, recall that for this particular use case, various configurations for on-line training were tried, with batch sizes of one. Although this is potentially one factor, a motivation for future experiments is to look at the effect of the dataset size for both architectures.

Further on, these observations with training times are not entirely surprising. LSTM includes multiple gating mechanisms requiring additional computational

effort. While one LSTM-unit has three different gates and an increased number of operations, a CNN layer only involves the convolution operation. The latter functions as a sliding window and enables faster computations. Although the networks implemented in this thesis are sparse, this further showcase the potential use case of CNNs for time series in terms of model efficiency.

4.4.2 The importance of data

During model training, we observed multiple outcomes that affected the model performance and learning the data distribution. First, one observation was related to particular outlier partitions in the training process during cross-validation. These were identified by higher validation loss and lower validation accuracy than usual in addition to lower evaluations compared to other partitions. Moreover, the training stopped far earlier than usual on these occasions. The problem was especially evident for the depression dataset in which it was shown how the models failed to capture multimodalities in the depression dataset. Another concern, however, is whether the selected input features are descriptive enough to learn the data distribution efficiently.

While the depression dataset is a univariate time-series with only one variable, the outliers were observed for the football use case as well, which includes multiple variables in the dataset. Although the data analysis did not present evidence of multimodalities in readiness scores, the problem of outlier partitions and input features is more probable for the football use case. Arguably, with less descriptive input features, it may pose increased confusion in the classification task. For the football use case, this was not explored extensively due to time limitations, but in future studies for similar domains, this should be taken into consideration. Lastly, for the EV use case, the problem of multimodality or selection of input features is difficult to determine. Because the dataset is relatively small for machine learning tasks, the results only showcase the potential improvement for future studies.

Nevertheless, to further understand the adaptability to the underlying data distribution, we can look at the number of training iterations used. **Figure 4.37** shows the average number of epochs for each experiment across all use cases. Overall, the figure shows that the LSTM uses less training iterations in learning all distributions. However, the number of iterations used depends on the dataset and differences are marginal for the football use case, although the pattern is the same. Notably, comparing this with the training time used in seconds, there is a trade-off between absolute time used and time used for learning the underlying distribution. Whether this holds for larger models and datasets is uncertain, but the results suggest that regardless of use case, training time and training iterations differs notably between CNN and LSTM. However, this should be tested for larger models and datasets as well, to understand further if the effect is proportional to dataset size and model size.

4.4.3 The effect of model architecture

On uni-dimensional data, the convolution operation in CNNs can be interpreted as a moving average, where the kernel size and dilation rate determine the total

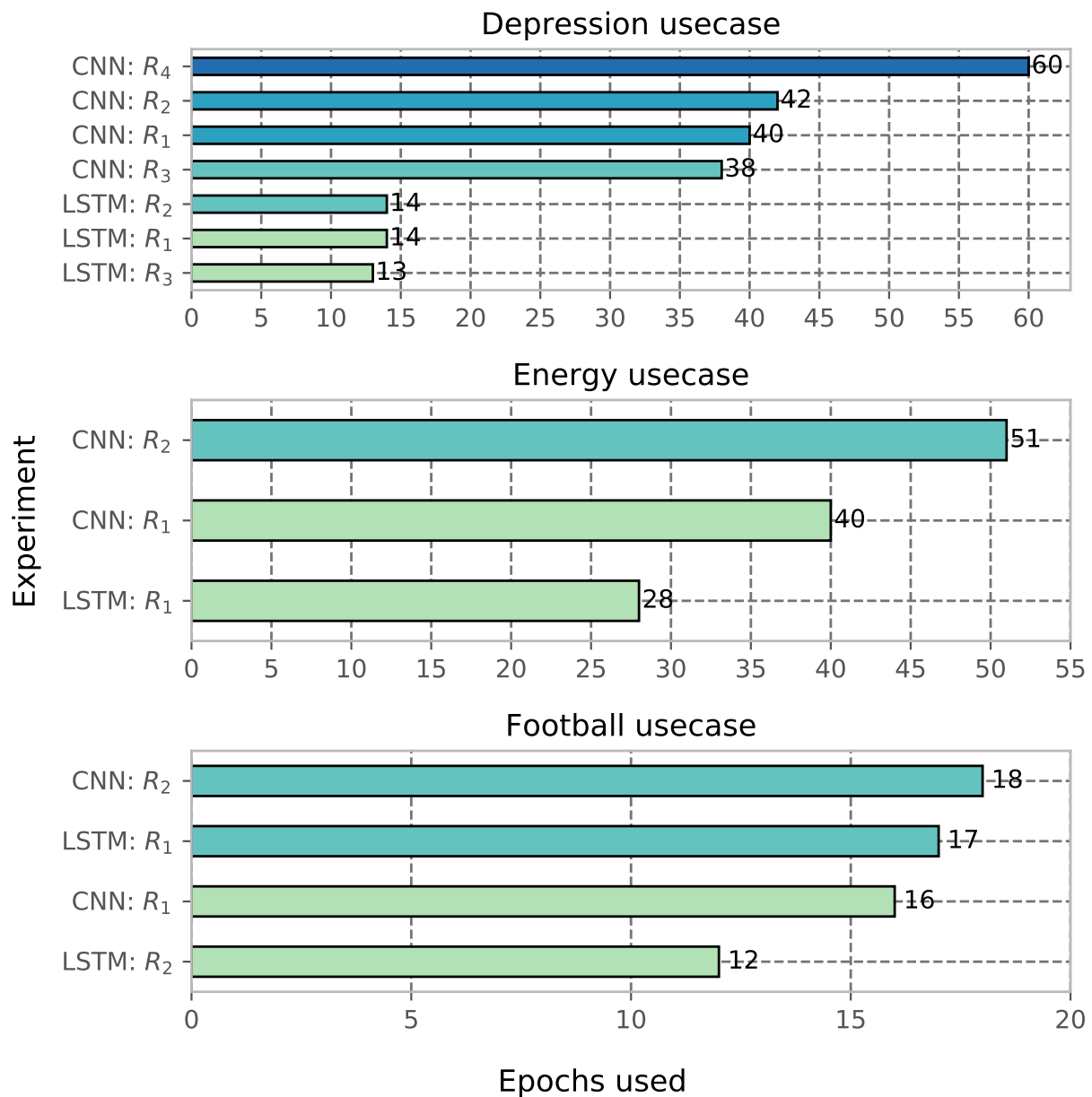


Figure 4.37: The figure illustrate average number of epochs used across cross-validation, for all experiments in all use cases. While the maximum training iterations are set to 300, both models terminates training earlier because of early stopping.

window size. In our findings, we observe how CNN performs as well as the LSTM across all use cases. Arguably, while hyperparameter configurations may explain one aspect of it, the effect of the sliding window mechanism in CNN can also explain its performance gains. Hypothetically, convolving the kernel across the input enables the CNN to capture localised patterns by only looking at a subset of the whole sequence. Compared to LSTM, which captures temporal dependencies through its gates, our CNN seems to do the same through the number of filters used, where each filter helps in detecting various localised patterns.

In general terms, it can be argued that the number of output units for each LSTM-

cell constrains the LSTM to what information can be exposed, because of its output gate. For the CNN, the same number of output units will thus expose all information in the output map, because there is no gated control. As a result of this, we see the performance is highly comparable to the LSTM and can draw certain parallels to the GRU-architecture, which provides the same mechanism. However, the hypothesis that CNN provides a similar feature is difficult to confirm without further experiments. We did not explore various configurations on the hidden layers, nodes in hidden layers (filters), kernel sizes and sequence windows extensively to sufficiently draw such a conclusion.

4.4.4 Application areas and use cases

The overall findings suggest that CNN should be the preferred model because of faster training and almost equal performance. However, in the context of each use case, the model selection should not only depend on this trade-off.

Depression detection - It is evident that multimodalities affect both models, but the LSTM is more robust in learning the data distribution. Overall the MCC is also higher compared to our CNN, indicating that LSTM is more robust in detecting various latent features. This suggests that there are underlying patterns in the dataset which CNN fails to capture equally well, in which we suggest LSTM should be used. In the case of depression classification, which is a critical task in mental health treatment, it is more important in having a robust model, rather than a model that can be trained faster. However, with a more diverse dataset or the addition of discriminative features, we believe CNN would achieve more robust results as well, and it should not be disregarded.

Energy prediction - The overall performance of both models are very similar. CNN is slightly better overall, with a positive MCC, but both models perform equally well across all use experiments. While the EV dataset is relatively small for machine learning tasks, it is difficult to determine the preferred approach for this domain. However, we strongly emphasise that CNN should be the preferred method for experimental methods, to gain faster knowledge on the effects of neural network modelling for time-series classification in EV energy optimisation. Overall, similar to the depression dataset, we believe a larger and more diverse dataset with more EVs can improve our baselines further. In that case, the effects of LSTM can be explored to optimise the classification task further, because, as we have seen, it is more robust in learning underlying data distributions.

Football readiness classification - We see no distinct trade-off in model performance. The LSTM is marginally better, although it achieves somewhat better MCC-scores. We evidently see that the trade-off in training time for CNN is beneficial, as it achieves almost the same results as LSTM. Arguably, CNN should be the preferred approach for classifying readiness of football players, but another topic of discussion is how variations in the reported scores could potentially improve the classification task for both models, particularly the specificity score. Recall the presented confusion matrices for our best performing models, where we saw how CNN had increased misclassifications of soreness and fatigue classes. We believe the misclassifications can be attributed to the absence in higher variations in reported scores for soreness

and fatigue. More specifically, we know when players feel fresh and great, but not the other way, when they feel sore and tired, which implies a particular class imbalance. In a way, these attributes can be captured by including additional descriptive features that explain soreness and fatigue better. Moreover, it may potentially also improve the specificity rate, hypothetically, if the model is good at finding the underlying patterns. Overall, our multilabel classification approach can help coaches to determine readiness based on multiple descriptive output features, which can be extended further to many categorical variables. To determine this effectively, however, we emphasise that additional features should be used, which are more discriminative towards each variable.

4.5 Summary and overview of all experiments

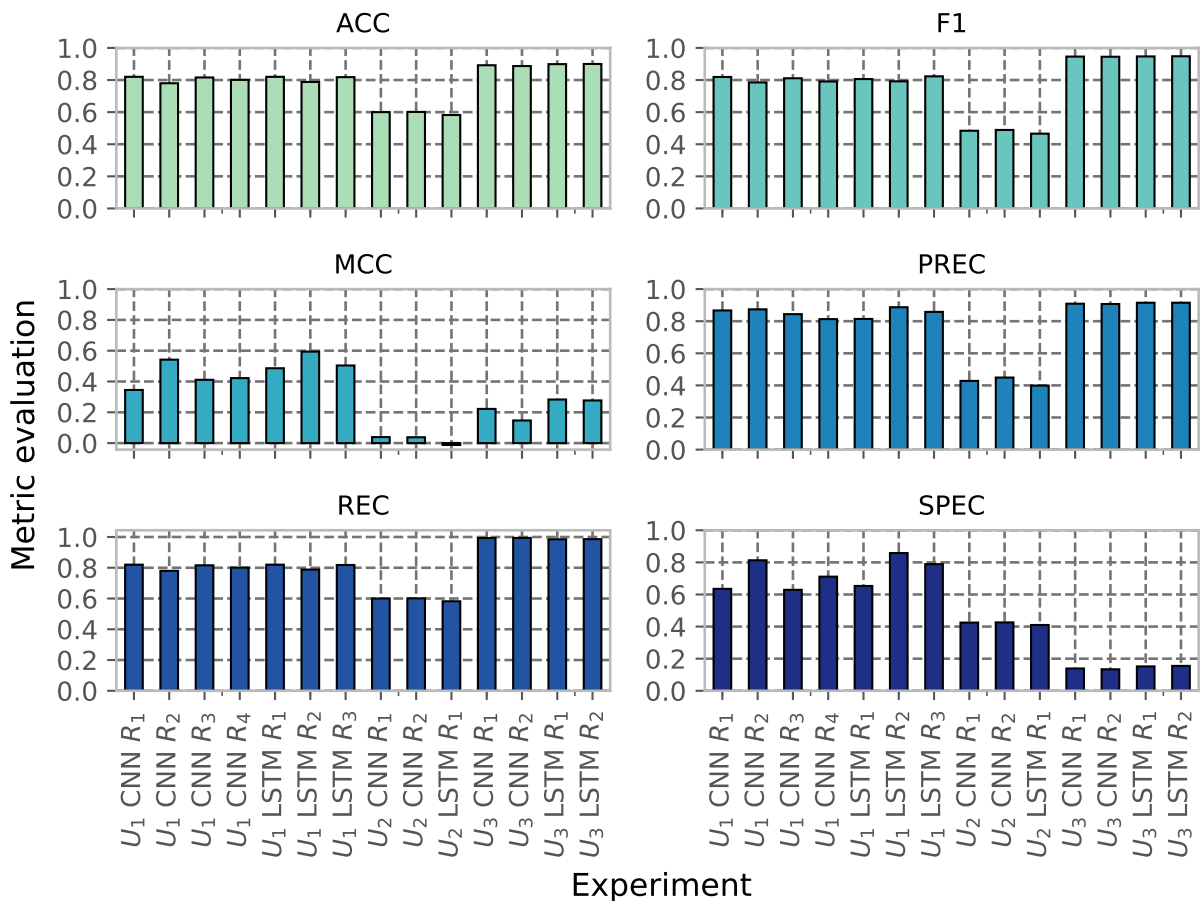


Figure 4.38: The figure illustrates a summary of all experiments in this thesis, with average metric evaluations for each experiment. The notation U_1 , U_2 and U_3 denotes the use cases depression, energy and football, respectively. A tabular overview of this figure is found in [Appendix B.1](#).

Figure 4.38 summarises all the experiments in this thesis. In this chapter, we first outlined a general level experiment design, presented and discussed the experiment evaluation methods. Further on, we discussed all experiments on a use case level.

For each use case, the background on the domain was discussed before doing data analysis. Moreover, we looked at the experiment design and discussed different choices made. Lastly, we presented the results for LSTM and CNN followed by a discussion of both models for each use case and a summary. Finally, a general level discussion was provided, based on the observations made across all experiments and use cases.

The results suggest that uni-dimensional CNNs can be applied to time-series classification and outperform LSTM, both in terms of predictive performance and total training time used. However, two concerns were drawn between both models. First, there is particularly difficult to avoid the curse of multimodality, emphasising that the importance of data quality is crucial for training. Second, there is a trade-off between the time used for training and the number of training iterations used in learning the data distribution. We see that despite CNN using less time to train, our sparse network uses more time to fit the data distribution efficiently. Hence, the CNN achieves marginally better results to the LSTM in many cases, but the latter terminates faster and learns better.

Furthermore, while various hyperparameter configurations were tried, it was seemingly difficult to optimise the models extensively. In many cases, throughout all experiments, the CNN was more sensitive to hyperparameters than the LSTM and achieved similar results across experiments. The LSTM, however, was more robust to different settings and was experienced as easier to train in this context, although time used was longer than the CNN.

Chapter 5

Conclusion

5.1 Summary

The aim of this thesis focused on comparing uni-dimensional CNN against LSTM for time-series classification, which is one type of sequential problems. The goal was to answer how CNNs can be applied to temporal domains like time-series classification because traditional methods are often used [2, 82]. In the domain of time-series modelling, the application of neural networks is not extensive. However, research in recent years show that both RNN-based models [28, 44, 83, 84] and CNN-based models [12, 28, 56, 78, 93] are applicable, but the comparative studies in the field are very limited.

To further understand the effects of CNN on time-series classification and provide a systematic comparative overview against LSTM, we performed multiple experiments across three use cases. First, we tried to classify depressed patients based on motor activity levels and reported results which improved existing baselines for both models. Second, we predicted energy demand for EVs and showed that there is a high potential for improvements. Our results are promising baselines for future studies. Third, we predicted readiness of football players, by formulating a multilabel classification problem. The results were promising, indicating the predictive systems that output a set of qualitative features could be used to understand the readiness of football players better. The overall process of performing these experiments was divided into a three-step process. We started with data analysis for each use case to form hypotheses that motivated for initial hyperparameter configurations. Further on, we ran initial experiments to observe how our models learned the data distributions and how they responded to various settings. At last, the most promising hyperparameter configurations were chosen to be explored further for model fine-tuning.

To perform these experiments, we implemented an experimental framework that automatically configures and builds our models, runs experiments and stores results. We used the Python programming language for implementation and the Keras framework for neural network model development. We developed it to be specific to our use case, but general enough to be extensible for future comparative studies. Additional modules or models can be integrated into our framework with minor

adjustments. Overall, we separated our system into three parts, namely configuration and initialisation, experiment pipeline and evaluation pipeline. In essence, the experiment pipeline enables auto-execution of a given experiment based on a configuration file in JSON-format. This includes data-formatting, model selection, cross-validation splitting, model building and compilation and lastly, training. Further on, the evaluation pipeline keeps track of training history, evaluations such as confusion matrices and metrics, various statistics, and saves this information to persistent storage along with the trained models and used configurations.

5.2 Contributions

Conclusively, we have developed an experimental framework as summarised above and presented a comparative study of LSTM and uni-dimensional CNN on three use cases, with their potential application areas in their specific domains. Lastly, we explored the following research question, as highlighted in **Section 1.2**:

RQ: How do CNNs compare to LSTMs for time-series classification?

Further on, recall that we derived two sub-questions to understand how both architectures perform in the context of our classification tasks:

1. *SQ1: How does LSTM perform for time-series classification?*
2. *SQ2: How does CNN perform for time-series classification?*

First, the application of LSTM to the time-series domain is not entirely unknown [28, 83]. We show that LSTM can be applied to various tasks in terms of time-series classification, further extending existing research. We classify depressed patients based on motor activity levels and achieve an accuracy of $82\% \pm 1.5\%$ across all experiments, whereas we report an MCC of 0.59 for the optimal experiment, improving existing baselines of 73% and 0.43, respectively. Further on, we achieve an averaged F1-score of 0.82, across all cross-evaluations, whereas the current baseline is 0.72. For the second use case, we classify energy demand for EVs, showcasing potential improvement areas for this particular task. Overall, our LSTM achieves an accuracy of 60% and an MCC of -0.01. We report precision of 0.40, a specificity of 0.41, 0.58 in recall and 0.40 in F1-score. The results are promising, considering how the dataset size is small and less diverse in terms of the number of EVs. For the last use case, on the other hand, we see a notable performance for multilabel classification tasks. We achieve accuracy scores of almost 90% and F1-scores of 0.95. The recall is closer to 0.99 for all experiments, but we see deviations in MCC and specificity, possibly related to imbalances in the PMSys dataset for reported scores.

Further on, CNN is arguably more efficient than LSTM and performs the classification tasks equally well in many experiments. For the depression use case, we report an accuracy of $80\% \pm 2\%$ across all experiments. Overall, CNN also improves the current baseline accuracy of 73%. The best reported MCC is 0.54, showing an improvement on the MCC baseline of 0.43. F1-scores are also consistent around 0.80 across all experiments. Further on, for the second use case with energy demand classification, the results are relatively similar to LSTM. We report accuracy, F1 and MCC to 60%, 0.49 and 0.04, respectively. Lastly, despite performing marginally lower

than LSTM for the football use case, we report accuracy scores and F1-scores of 89% and 0.95, respectively, whereas the best MCC is 0.22 and the worst is 0.15 for our two optimised experiments. Overall, we achieve the same performance as LSTM and experience the CNN as much faster to train. CNN is efficiently applicable to the time-series classification domain and should be the preferred approach for faster training.

Overall, to answer our research question, our findings can be attributed to three important aspects of this comparative study. First, our results suggest that CNN performs the classification tasks equally well as LSTM, or better at its best, regardless of use case. Second, what is more notable, is that CNN is faster to train for two of three use cases. For classifying depressed patients, it is on average 46 times faster than LSTM when comparing seconds used for training, whereas for classification of football readiness, CNN is 35 times faster. Lastly, while this efficiency in training times is distinct, we see that LSTM is more robust and learns underlying data distributions faster. CNN uses more training iterations and converges slower. The variations are minimal, but at its best, the LSTM is three times more efficient on average in terms of the number of training iterations used. Comparably, the speedup factor for CNN in terms of training time is notably higher, as opposed to the speedup factor in training iterations for LSTM.

5.3 Future work

Preferably, we would like to look at five different improvement areas. First, to improve model performance and optimisation, the model fine-tuning should resort to automatic hyperparameter optimisation methods. Second, other architectures should be explored and compared, like ResNet and GRU, which should provide better comparative baselines. Third, the effect of dataset size and variation was notable in many experiments, and for future work, we would like to include larger and diverse datasets and also evaluate against other time-series datasets. Fourth, we would like to explore the effects of CNN-specific architecture elements through ablation studies and analyse the effect of the convolution operation, dilated convolutions and different kernel sizes. At last, an interesting aspect we want to try for future work is to represent time-series as two-dimensional topologies and look at the effect of CNNs from another perspective with state-of-the-art CNNs and transfer learning.

Experiment with automatic hyperparameter optimisation

We suggest that future studies should use automatic hyperparameter optimisation to fine-tune the models. Model optimisation and fine-tuning were manual in our comparisons and an overall tedious task. With automatic optimisation of data-related and network-related hyperparameters, the performance of the models can potentially be improved. We observed that across all use cases, the importance of data affected model performance. Hence, finding optimal data-related hyperparameters could be difficult as well, as it depends on the variations and size of the dataset. Overall, these hyperparameters can be explored through automatic hyperparameter optimisation methods, like Bayesian Optimisation, Grid Search or Random Search.

Experiment with deeper networks and other architectures

We limited our study to a comparison of shallow LSTM and CNN and showed promising empirical evidence for CNN as a preferred architecture over LSTM for time-series classification. We emphasise that more complex and potentially deeper models should be tested to evaluate whether our findings hold for larger models. Additionally, other architectures should be explored as well. One approach is to explore different architectures for LSTM and CNN through methods such as Neural Architecture Search (NAS) [96]. Various other models should also be tried, like GRU or ResNet, to understand their effects.

Train with larger, diverse datasets and other datasets

While we tested on three different datasets, we experienced curiosities that affected model performance in various ways. We observed multimodality and outliers, bad model convergence and misclassifications between specific labels in various experiments. Arguably, researchers should improve these problems by including increased variations, more qualitative and discriminative features and in general, larger datasets. Although data itself is one shortcoming, these suggestions will potentially strengthen future comparative studies and possibly prevent biased models as well. Furthermore, LSTM and CNN should also be evaluated on various other time-series datasets as well, to provide stronger baselines and comparisons.

Comparing architecture elements and conducting ablation studies

In our discussions, we analysed training times, training history, model convergence and effect of various hyperparameters. However, one part left out of our study is the effect of CNN-specific elements. For instance, we did not perform extensive studies on convolutions, dilated convolutions or explore the effects of pooling layers. Although we tried different dilation rates and kernel sizes, it is uncertain whether these had any significant impact on the model performance. Future work should look into architecture-specific elements or perform ablation studies [60]. This includes analysing the effects of convolutions, various kernel sizes and effect of dilated convolutions in CNNs. Not only is this important in understanding how CNNs can be preferable over LSTM, or RNNs in general, but their underlying and most important mechanism, the convolution operation, can be understood in detail in the context of time-series analysis.

Two-dimensional CNNs and transfer learning

While we explored the effect of uni-dimensional CNN, there are no existing state-of-the-art models in the field of time-series classification, similar to that in the field of computer vision. For future work, an interesting aspect is to encode time-series as images, which enables the possibility of applying state-of-the-art computer vision algorithms and transfer learning.

Bibliography

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. “TensorFlow: A system for large-scale machine learning”. In: *arXiv:1605.08695 [cs]* (May 27, 2016). arXiv: 1605.08695. URL: <http://arxiv.org/abs/1605.08695> (visited on 04/03/2019).
- [2] Ratnadip Adhikari and R. K. Agrawal. “An Introductory Study on Time Series Modeling and Forecasting”. In: *arXiv:1302.6613 [cs, stat]* (Feb. 26, 2013). arXiv: 1302.6613. URL: <http://arxiv.org/abs/1302.6613> (visited on 01/18/2019).
- [3] Tuka Al Hanai, Mohammad Ghassemi, and James Glass. “Detecting Depression with Audio/Text Sequence Modeling of Interviews”. In: *Interspeech 2018*. Interspeech 2018. ISCA, Sept. 2, 2018, pp. 1716–1720. DOI: 10.21437/Interspeech.2018-2522. URL: http://www.isca-speech.org/archive/Interspeech_2018/abstracts/2522.html (visited on 02/04/2019).
- [4] Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C. Van Esesn, Abdul A. S. Awwal, and Vijayan K. Asari. “The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches”. In: *arXiv:1803.01164 [cs]* (Mar. 3, 2018). arXiv: 1803.01164. URL: <http://arxiv.org/abs/1803.01164> (visited on 01/17/2019).
- [5] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling”. In: *arXiv:1803.01271 [cs]* (Mar. 3, 2018). arXiv: 1803.01271. URL: <http://arxiv.org/abs/1803.01271> (visited on 01/19/2019).
- [6] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. “ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst”. In: *arXiv:1812.03079 [cs]* (Dec. 7, 2018). arXiv: 1812.03079. URL: <http://arxiv.org/abs/1812.03079> (visited on 01/17/2019).
- [7] *Barca Innovation Hub*. Barca Innovation Hub. URL: <https://barcainnovationhub.com/> (visited on 09/19/2019).
- [8] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (Mar. 1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181.

-
- [9] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *arXiv:1206.5533 [cs]* (June 24, 2012). arXiv: 1206.5533. URL: <http://arxiv.org/abs/1206.5533> (visited on 10/20/2019).
- [10] Jan O Berle, Erik R Hauge, Ketil J Oedegaard, Fred Holsten, and Ole B Fasmer. "Actigraphic registration of motor activity reveals a more structured behavioural pattern in schizophrenia than in major depression". In: *BMC Research Notes* 3.1 (2010), p. 149. ISSN: 1756-0500. DOI: 10.1186/1756-0500-3-149. URL: <http://bmcresnotes.biomedcentral.com/articles/10.1186/1756-0500-3-149> (visited on 01/16/2019).
- [11] Rune Borgli, Pål Halvorsen, Michael Riegler, and Håkon Stensland. *Automatic Hyperparameter Optimization in Keras for the MediaEval 2018 Medico Multimedia Task*. 2018. URL: <https://www.simula.no/publications/automatic-hyperparameter-optimization-keras-mediaeval-2018-medico-multimedia-task>.
- [12] Anastasia Borovykh, Sander Bohte, and Cornelis W. Oosterlee. "Conditional Time Series Forecasting with Convolutional Neural Networks". In: *arXiv:1703.04691 [stat]* (Mar. 14, 2017). arXiv: 1703.04691. URL: <http://arxiv.org/abs/1703.04691> (visited on 03/24/2019).
- [13] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. "cuDNN: Efficient Primitives for Deep Learning". In: *arXiv:1410.0759 [cs]* (Oct. 3, 2014). arXiv: 1410.0759. URL: <http://arxiv.org/abs/1410.0759> (visited on 04/08/2019).
- [14] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *arXiv:1406.1078 [cs, stat]* (June 3, 2014). arXiv: 1406.1078. URL: <http://arxiv.org/abs/1406.1078> (visited on 05/05/2018).
- [15] François Chollet. *Keras - Neural network library*. Keras: The Python Deep Learning library. URL: <https://keras.io/> (visited on 04/03/2019).
- [16] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *arXiv:1412.3555 [cs]* (Dec. 11, 2014). arXiv: 1412.3555. URL: <http://arxiv.org/abs/1412.3555> (visited on 05/29/2018).
- [17] Razvan-Gabriel Cirstea, Darius-Valer Micu, Gabriel-Marcel Muresan, Chenjuan Guo, and Bin Yang. "Correlated Time Series Forecasting using Deep Neural Networks: A Summary of Results". In: *arXiv:1808.09794 [cs, stat]* (Aug. 29, 2018). arXiv: 1808.09794. URL: <http://arxiv.org/abs/1808.09794> (visited on 03/25/2019).
- [18] Matthew Conlen. *Kernel Density Estimation*. URL: <https://mathisonian.github.io/kde/> (visited on 05/11/2019).
- [19] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: <http://cs231n.github.io/linear-classify/#svmvssoftmax> (visited on 01/23/2019).
- [20] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: <http://cs231n.github.io/neural-networks-2/#datapre> (visited on 05/15/2019).
-

- [21] Peter J. Denning, D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. "Computing as a discipline". In: *Communications of the ACM* 32.1 (Feb. 1, 1989), pp. 9–23. ISSN: 00010782. DOI: 10.1145/63238.63239. URL: <http://portal.acm.org/citation.cfm?doid=63238.63239> (visited on 02/21/2019).
- [22] *Depression*. URL: <https://www.who.int/news-room/fact-sheets/detail/depression> (visited on 02/04/2019).
- [23] *Depression in adults: recognition and management | Guidance and guidelines | NICE*. URL: <https://www.nice.org.uk/guidance/cg90/chapter/Introduction> (visited on 02/04/2019).
- [24] Maha Elbayad, Laurent Besacier, and Jakob Verbeek. "Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction". In: *arXiv:1808.03867 [cs]* (Aug. 11, 2018). arXiv: 1808.03867. URL: <http://arxiv.org/abs/1808.03867> (visited on 11/08/2018).
- [25] *Energy market data*. Energy market data. URL: <http://www.nordpoolspot.com/Market-data1/> (visited on 01/19/2019).
- [26] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller. "Deep learning for time series classification: a review". In: *Data Mining and Knowledge Discovery* (Mar. 2, 2019). ISSN: 1384-5810, 1573-756X. DOI: 10.1007/s10618-019-00619-1. arXiv: 1809.04356. URL: <http://arxiv.org/abs/1809.04356> (visited on 03/25/2019).
- [27] *ForzaSys - pmSys*. URL: <https://forzasys.com/pmsys.html> (visited on 09/17/2019).
- [28] John Cristian Borges Gamboa. "Deep Learning for Time-Series Analysis". In: *arXiv:1701.01887 [cs]* (Jan. 7, 2017). arXiv: 1701.01887. URL: <http://arxiv.org/abs/1701.01887> (visited on 03/25/2019).
- [29] Enrique Garcia-Ceja, Michael Riegler, Petter Jakobsen, Jim Tørresen, Tine Nordgreen, Ketil J. Oedegaard, and Ole Bernt Fasmer. "Depresjon: a motor activity database of depression episodes in unipolar and bipolar patients". In: *Proceedings of the 9th ACM Multimedia Systems Conference on - MMSys '18*. the 9th ACM Multimedia Systems Conference. MMSys'18. Amsterdam, Netherlands: ACM Press, 2018, pp. 472–477. ISBN: 978-1-4503-5192-8. DOI: 10.1145/3204949.3208125. URL: <http://dl.acm.org/citation.cfm?doid=3204949.3208125> (visited on 01/16/2019).
- [30] *Gartner Says Deep Learning Will Provide Best-in-Class Performance for Demand, Fraud and Failure Predictions By 2019*. URL: <https://www.gartner.com/en/newsroom/press-releases/2017-09-20-gartner-says-deep-learning-will-provide-best-in-class-performance-for-demand-fraud-and-failure-predictions-by-2019> (visited on 01/21/2019).
- [31] *GEVO 2018*. URL: <https://www.iea.org/gevo2018/> (visited on 01/24/2019).
- [32] *Global EV Outlook 2018*. IEA webstore. URL: <https://webstore.iea.org/global-ev-outlook-2018> (visited on 12/05/2018).
- [33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.

- [34] Alex Graves. “Generating Sequences With Recurrent Neural Networks”. In: *arXiv:1308.0850 [cs]* (Aug. 4, 2013). arXiv: 1308.0850. URL: <http://arxiv.org/abs/1308.0850> (visited on 11/24/2018).
- [35] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech Recognition with Deep Recurrent Neural Networks”. In: *arXiv:1303.5778 [cs]* (Mar. 22, 2013). arXiv: 1303.5778. URL: <http://arxiv.org/abs/1303.5778> (visited on 05/05/2018).
- [36] *Hafslund Case Study*. Google Cloud. URL: <https://cloud.google.com/customers/hafslund/> (visited on 12/05/2018).
- [37] Hossein Hassani. “Research Methods in Computer Science: The Challenges and Issues”. In: *arXiv:1703.04080 [cs]* (Mar. 12, 2017). arXiv: 1703.04080. URL: <http://arxiv.org/abs/1703.04080> (visited on 02/21/2019).
- [38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *arXiv:1512.03385 [cs]* (Dec. 10, 2015). arXiv: 1512.03385. URL: <http://arxiv.org/abs/1512.03385> (visited on 03/24/2019).
- [39] S. Hicks, M. Riegler, K. Pogorelov, K. V. Anonsen, T. de Lange, D. Johansen, M. Jeppsson, K. Ranheim Randel, S. Losada Eskeland, and P. Halvorsen. “Dissecting Deep Neural Networks for Better Medical Image Classification and Classification Understanding”. In: *2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS)*. 2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS). June 2018, pp. 363–368. DOI: 10.1109/CBMS.2018.00070.
- [40] Sepp Hochreiter. “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06.2 (Apr. 1998), pp. 107–116. ISSN: 0218-4885, 1793-6411. DOI: 10.1142/S0218488598000094. URL: <http://www.worldscientific.com/doi/abs/10.1142/S0218488598000094> (visited on 03/07/2019).
- [41] Sepp Hochreiter and Jürgen Schmidhuber. “LONG SHORT-TERM MEMORY”. In: *MIT Press Cambridge, MA, USA* (Nov. 15, 1997). URL: <http://www.bioinf.jku.at/publications/older/2604.pdf> (visited on 05/05/2018).
- [42] J. J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: *Proceedings of the National Academy of Sciences* 79.8 (Apr. 1, 1982), pp. 2554–2558. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.79.8.2554. URL: <https://www.pnas.org/content/79/8/2554> (visited on 01/21/2019).
- [43] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. “Squeeze-and-Excitation Networks”. In: *arXiv:1709.01507 [cs]* (Sept. 5, 2017). arXiv: 1709.01507. URL: <http://arxiv.org/abs/1709.01507> (visited on 03/24/2019).
- [44] Michael Hüsken and Peter Stagge. “Recurrent neural networks for time series classification”. In: *Neurocomputing* 50 (Jan. 2003), pp. 223–235. ISSN: 09252312. DOI: 10.1016/S0925-2312(01)00706-8. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0925231201007068> (visited on 04/01/2019).

- [45] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, eds. *An introduction to statistical learning: with applications in R*. Springer texts in statistics 103. OCLC: ocn828488009. New York: Springer, 2013. 426 pp. ISBN: 978-1-4614-7137-0.
- [46] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An Empirical Exploration of Recurrent Network Architectures". In: *Journal of Machine Learning Research* (), p. 9. URL: <http://proceedings.mlr.press/v37/jozefowicz15.pdf>.
- [47] Daniel Jurafsky and James H. Martin. "Hidden Markov Models". In: *Speech and Language Processing*. 3rd ed. 2018. URL: <https://web.stanford.edu/~jurafsky/slp3/>.
- [48] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. "Neural Machine Translation in Linear Time". In: *arXiv:1610.10099 [cs]* (Oct. 31, 2016). arXiv: 1610.10099. URL: <http://arxiv.org/abs/1610.10099> (visited on 01/19/2019).
- [49] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A Convolutional Neural Network for Modelling Sentences". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 655–665. DOI: 10.3115/v1/P14-1062. URL: <http://aclweb.org/anthology/P14-1062> (visited on 01/19/2019).
- [50] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Dec. 22, 2014). arXiv: 1412.6980. URL: <http://arxiv.org/abs/1412.6980> (visited on 02/02/2019).
- [51] Tarjei Kristiansen. "A time series spot price forecast model for the Nord Pool market". In: *International Journal of Electrical Power & Energy Systems* 61 (Oct. 2014), pp. 20–26. ISSN: 01420615. DOI: 10.1016/j.ijepes.2014.03.007. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0142061514001094> (visited on 04/24/2018).
- [52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (May 24, 2017), pp. 84–90. ISSN: 00010782. DOI: 10.1145/3065386. URL: <http://dl.acm.org/citation.cfm?doid=3098997.3065386> (visited on 05/10/2018).
- [53] Simon Kuper. *How FC Barcelona are preparing for the future of football*. Financial Times. Mar. 1, 2019. URL: <https://www.ft.com/content/908752aa-3a1b-11e9-b72b-2c7f526ca5d0> (visited on 09/19/2019).
- [54] *Learning to Drive by Imitating the Best and Synthesizing the Worst*. Waymo Research. URL: <https://sites.google.com/view/waymo-learn-to-drive> (visited on 01/21/2019).
- [55] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.
- [56] C. Liu, W. Hsaio, and Y. Tu. "Time Series Classification With Multivariate Convolutional Neural Network". In: *IEEE Transactions on Industrial Electronics* 66.6 (June 2019), pp. 4788–4797. ISSN: 0278-0046. DOI: 10.1109/TIE.2018.2864702.

- [57] James Martens and Ilya Sutskever. “Learning Recurrent Neural Networks with Hessian-Free Optimization”. In: *Proceedings of the 28th International Conference on Machine Learning* (Jan. 2011), pp. 1033–1040. URL: http://www.icml-2011.org/papers/532_icmlpaper.pdf.
- [58] Warren McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), p. 21. URL: <http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf>.
- [59] Wes McKinney. “Pandas: a Foundational Python Library for Data Analysis and Statistics”. In: *Python High Performance Science Computer* (Jan. 2011), p. 9. URL: https://www.dlr.de/sc/Portaldata/15/Resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf.
- [60] Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. “Ablation Studies in Artificial Neural Networks”. In: *arXiv:1901.08644 [cs, q-bio]* (Feb. 18, 2019). arXiv: 1901.08644. URL: <http://arxiv.org/abs/1901.08644> (visited on 11/06/2019).
- [61] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems* 26 (Oct. 2013), p. 9. URL: <https://arxiv.org/abs/1310.4546v1>.
- [62] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. “Scalable Parallel Programming with CUDA”. In: *Queue* 6.2 (Mar. 2008), pp. 40–53. ISSN: 1542-7730. DOI: 10.1145/1365490.1365500. URL: <http://doi.acm.org/10.1145/1365490.1365500> (visited on 04/08/2019).
- [63] NIMH » *Bipolar Disorder*. URL: <https://www.nimh.nih.gov/health/topics/bipolar-disorder/index.shtml> (visited on 02/04/2019).
- [64] NIMH » *Depression*. URL: <https://www.nimh.nih.gov/health/topics/depression/index.shtml> (visited on 02/04/2019).
- [65] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. In: *arXiv:1811.03378 [cs]* (Nov. 8, 2018). arXiv: 1811.03378. URL: <http://arxiv.org/abs/1811.03378> (visited on 01/23/2019).
- [66] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. “WaveNet: A Generative Model for Raw Audio”. In: *arXiv:1609.03499 [cs]* (Sept. 12, 2016). arXiv: 1609.03499. URL: <http://arxiv.org/abs/1609.03499> (visited on 11/12/2018).
- [67] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training Recurrent Neural Networks”. In: *arXiv:1211.5063 [cs]* (Nov. 21, 2012). arXiv: 1211.5063. URL: <http://arxiv.org/abs/1211.5063> (visited on 05/05/2018).
- [68] Dabal Pedomonti. “Comparison of non-linear activation functions for deep neural networks on MNIST classification task”. In: *arXiv:1804.02763 [cs, stat]* (Apr. 8, 2018). arXiv: 1804.02763. URL: <http://arxiv.org/abs/1804.02763> (visited on 01/23/2019).

- [69] Svein A. Pettersen, Håvard D. Johansen, Ivan A. M. Baptista, Pål Halvorsen, and Dag Johansen. “Quantified Soccer Using Positional Data: A Case Study”. In: *Frontiers in Physiology* 9 (2018). ISSN: 1664-042X. DOI: 10.3389/fphys.2018.00866. URL: <https://www.frontiersin.org/articles/10.3389/fphys.2018.00866/full> (visited on 11/03/2019).
- [70] Konstantin Pogorelov, Michael Riegler, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Carsten Griwodz, Peter Thelin Schmidt, and Pål Halvorsen. “Efficient disease detection in gastrointestinal videos – global features versus neural networks”. In: *Multimedia Tools and Applications* 76.21 (Nov. 1, 2017), pp. 22493–22525. ISSN: 1573-7721. DOI: 10.1007/s11042-017-4989-y. URL: <https://doi.org/10.1007/s11042-017-4989-y> (visited on 04/03/2019).
- [71] *Python - Programming language*. Python.org. URL: <https://www.python.org/> (visited on 04/03/2019).
- [72] *RECOVERY NUTRITION FOR FOOTBALL: BUILDING A TEAM*. Barça Innovation Hub. Sept. 24, 2018. URL: <https://barcainnovationhub.com/recovery-nutrition-for-football/> (visited on 09/29/2019).
- [73] *Renewable Energy Statistics 2018*. /publications/2018/Jul/Renewable-Energy-Statistics-2018. URL: /publications/2018/Jul/Renewable-Energy-Statistics-2018 (visited on 12/05/2018).
- [74] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519. URL: <http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519> (visited on 01/23/2019).
- [75] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv:1609.04747 [cs]* (Sept. 15, 2016). arXiv: 1609.04747. URL: <http://arxiv.org/abs/1609.04747> (visited on 01/31/2019).
- [76] D E Rumelhart, G E Hinton, and R J Williams. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. Vol. 1. MIT Press, 1985, pp. 318–362. ISBN: 0-262-68053-X. URL: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf>.
- [77] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge”. In: *arXiv:1409.0575 [cs]* (Sept. 1, 2014). arXiv: 1409.0575. URL: <http://arxiv.org/abs/1409.0575> (visited on 02/13/2019).
- [78] Lamyaa Sadouk. “CNN Approaches for Time Series classification”. In: *Time Series Analysis* (Nov. 5, 2018). DOI: 10.5772/intechopen.81170. URL: <https://www.intechopen.com/online-first/cnn-approaches-for-time-series-classification> (visited on 04/01/2019).
- [79] Haşim Sak, Andrew Senior, Kanishka Rao, and Françoise Beaufays. “Fast and Accurate Recurrent Neural Network Acoustic Models for Speech Recognition”. In: *arXiv:1507.06947 [cs, stat]* (July 24, 2015). arXiv: 1507.06947. URL: <http://arxiv.org/abs/1507.06947> (visited on 05/05/2018).

- [80] A. L. Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of Research and Development* 44.1 (Jan. 2000), pp. 206–226. ISSN: 0018-8646. DOI: 10.1147/rd.441.0206.
- [81] Bruce Schoenfeld. "How Data (and Some Breathtaking Soccer) Brought Liverpool to the Cusp of Glory". In: *The New York Times* (May 22, 2019). ISSN: 0362-4331. URL: <https://www.nytimes.com/2019/05/22/magazine/soccer-data-liverpool.html> (visited on 09/29/2019).
- [82] Robert H. Shumway and David S. Stoffer. *Time series analysis and its applications: with R examples*. 3rd ed. Springer texts in statistics. New York: Springer, 2011. 596 pp. ISBN: 978-1-4419-7864-6.
- [83] Sima Siami-Namini and Akbar Siami Namin. "Forecasting Economics and Financial Time Series: ARIMA vs. LSTM". In: *arXiv:1803.06386 [cs, stat]* (Mar. 16, 2018). arXiv: 1803.06386. URL: <http://arxiv.org/abs/1803.06386> (visited on 01/19/2019).
- [84] Denis Smirnov and Engelbert Mephu Nguifo. "Time Series Classification with Recurrent Neural Networks". In: *Advanced Analytics and Learning on Temporal Data* (2018), p. 8. URL: https://project.inria.fr/aaltd18/files/2018/08/aaltd18_rnn.pdf.
- [85] Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Marius Tennøe, Espen Helgedagsrud, Mikkel Næss, Henrik Kjus Alstad, Asgeir Mortensen, Ragnar Langseth, Sigurd Ljødal, Øystein Landsverk, Carsten Griwodz, Magnus Stenhaug, and Dag Johansen. "Bagadus: An Integrated Real-Time System for Soccer". In: *ACM Transactions on Multimedia Computing, Communications and Applications* (Jan. 2014), 41:1–41:21. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.455.884&rep=rep1&type=pdf>.
- [86] Olle Sundstrom and Carl Binding. "Flexible Charging Optimization for Electric Vehicles Considering Distribution Grid Constraints". In: *IEEE Transactions on Smart Grid* 3.1 (Mar. 2012), pp. 26–37. ISSN: 1949-3053, 1949-3061. DOI: 10.1109/TSG.2011.2168431. URL: <http://ieeexplore.ieee.org/document/6112699/> (visited on 05/10/2018).
- [87] Olle Sundstrom and Carl Binding. "Optimization Methods to Plan the Charging of Electric Vehicle Fleets". In: *Proceedings of the international conference on control, communication and power engineering* (July 28, 2010), pp. 28–29. URL: https://www.zurich.ibm.com/pdf/csc/EDISON_ccpe_main.pdf.
- [88] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions". In: *arXiv:1409.4842 [cs]* (Sept. 16, 2014). arXiv: 1409.4842. URL: <http://arxiv.org/abs/1409.4842> (visited on 03/24/2019).
- [89] *Tensorflow: An Open Source Machine Learning Framework for Everyone: tensorflow/tensorflow*. original-date: 2015-11-07T01:19:20Z. Apr. 3, 2019. URL: <https://github.com/tensorflow/tensorflow> (visited on 04/03/2019).
- [90] *TensorFlow - Machine learning platform*. An end-to-end open source machine learning platform. URL: <https://www.tensorflow.org/> (visited on 04/03/2019).

- [91] *The Depresjon Dataset*. The Depresjon Dataset. URL: <http://datasets.simula.no/depresjon/> (visited on 01/16/2019).
- [92] *The Financial Times highlights Barça's commitment to innovation*. URL: <https://www.fcbarcelona.com/en/news/1082842/the-financial-times-highlights-barcas-commitment-to-innovation> (visited on 09/19/2019).
- [93] Zhiguang Wang, Weizhong Yan, and Tim Oates. "Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline". In: *arXiv:1611.06455 [cs, stat]* (Nov. 19, 2016). arXiv: 1611.06455. URL: <http://arxiv.org/abs/1611.06455> (visited on 03/25/2019).
- [94] Theodor Wiik, Dag Johansen, Michael Riegler, Håvard Johansen, Svein-Arne Pettersen, Ivan Baptista, Tomas Kupa, and Pål Halvorsen. "Predicting Peek Readiness-to-Train of Soccer Players Using Long Short-Term Memory Recurrent Neural Networks". In: *Content-Based Multimedia Indexing (CBMI)* (Sept. 2019), p. 6. DOI: 10.1109/CBMI.2019.8877406. URL: <http://folk.uio.no/paalh/publications/files/cbmi2019-PMSYS.pdf>.
- [95] D.Randall Wilson and Tony R. Martinez. "The general inefficiency of batch training for gradient descent learning". In: *Neural Networks* 16.10 (Dec. 2003), pp. 1429–1451. ISSN: 08936080. DOI: 10.1016/S0893-6080(03)00138-2. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0893608003001382> (visited on 01/31/2019).
- [96] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. "A Survey on Neural Architecture Search". In: *arXiv:1905.01392 [cs, stat]* (June 18, 2019). arXiv: 1905.01392. URL: <http://arxiv.org/abs/1905.01392> (visited on 11/06/2019).
- [97] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *arXiv:1609.08144 [cs]* (Sept. 26, 2016). arXiv: 1609.08144. URL: <http://arxiv.org/abs/1609.08144> (visited on 02/14/2019).
- [98] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. "Comparative Study of CNN and RNN for Natural Language Processing". In: *arXiv:1702.01923 [cs]* (Feb. 7, 2017). arXiv: 1702.01923. URL: <http://arxiv.org/abs/1702.01923> (visited on 10/15/2019).

Appendices

Appendix A

Background

Appendices related to the background chapter.

A.1 Overfitting in machine learning

The appendix gives a simply illustration of the concept of overfitting in machine learning and statistical learning in general.

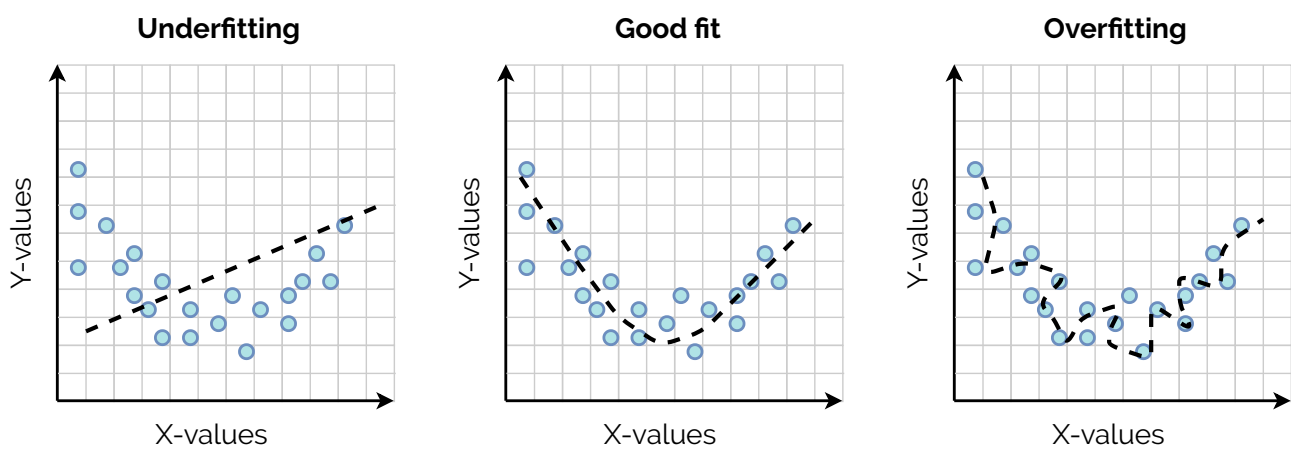


Figure A.1: Illustration of the concept of overfitting. The example shows how a function is fit to a set of observations, where the fit is either too generalised (underfitting), just right (good fit) and too specific (overfitting). Although the figure shows this for a simple case with X and Y , it is conceptually same for multi-dimensional spaces with many variables.

Appendix B

Experiments

Appendices related to the experiment chapter.

B.1 Depression: second initial CNN result

Appendix B.1.1, the first appendix shows a moving average of one of the initial results, which showed promising training progression. Further on, **Appendix B.1.2** show the confusion matrix for the best performing model.

B.1.1 Moving average of training history

B.1.2 Confusion matrix for best performing model

B.2 LSTM: additional experiments in energy prediction use case

The appendix shows additional LSTM experiments that were performed in the energy prediction use case, with various other configurations that were explored but not presented.

B.3 CNN: additional experiments in energy prediction use case

The appendix shows additional CNN experiments in the energy prediction use case, which were not presented in the thesis, but explored with various other configurations.

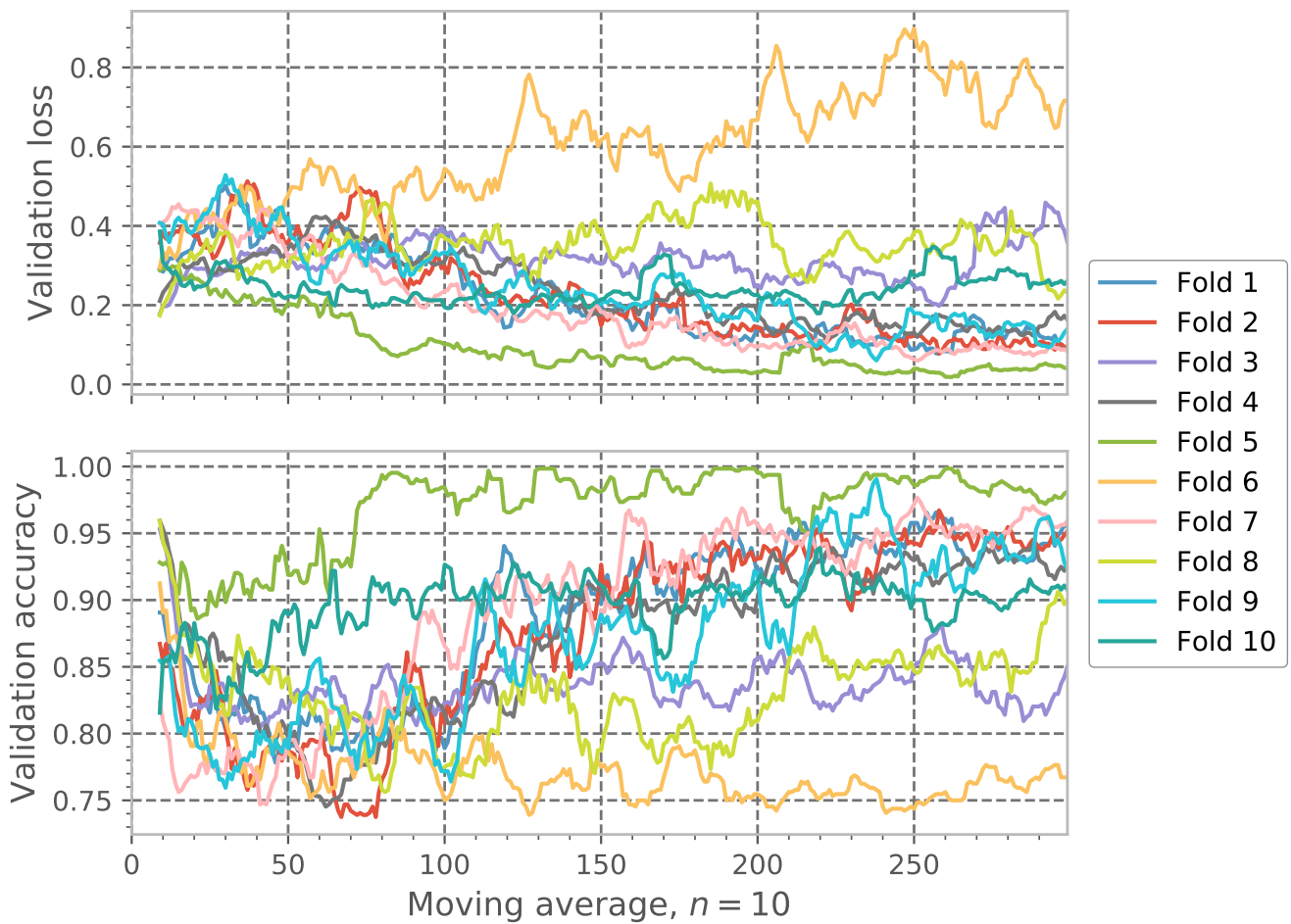


Figure B.1: The figure shows a 10-window moving average of the training history. Although being one of the initial results, the outcome shows that many of the folds converged well, both in terms of validation loss and accuracy, with promising results. The best performing model in this case, which did not overfit, achieved an accuracy of 0.91 and MCC of 0.84.

B.4 CNN: additional experiments in football use case

The appendix gives an overview over three additional CNN experiments conducted for the football use case, where different optimisers, sequence window lengths and kernel sizes were tried.

B.5 Table summary of all experiments

The appendix gives a tabular overview of all experiments in the thesis.

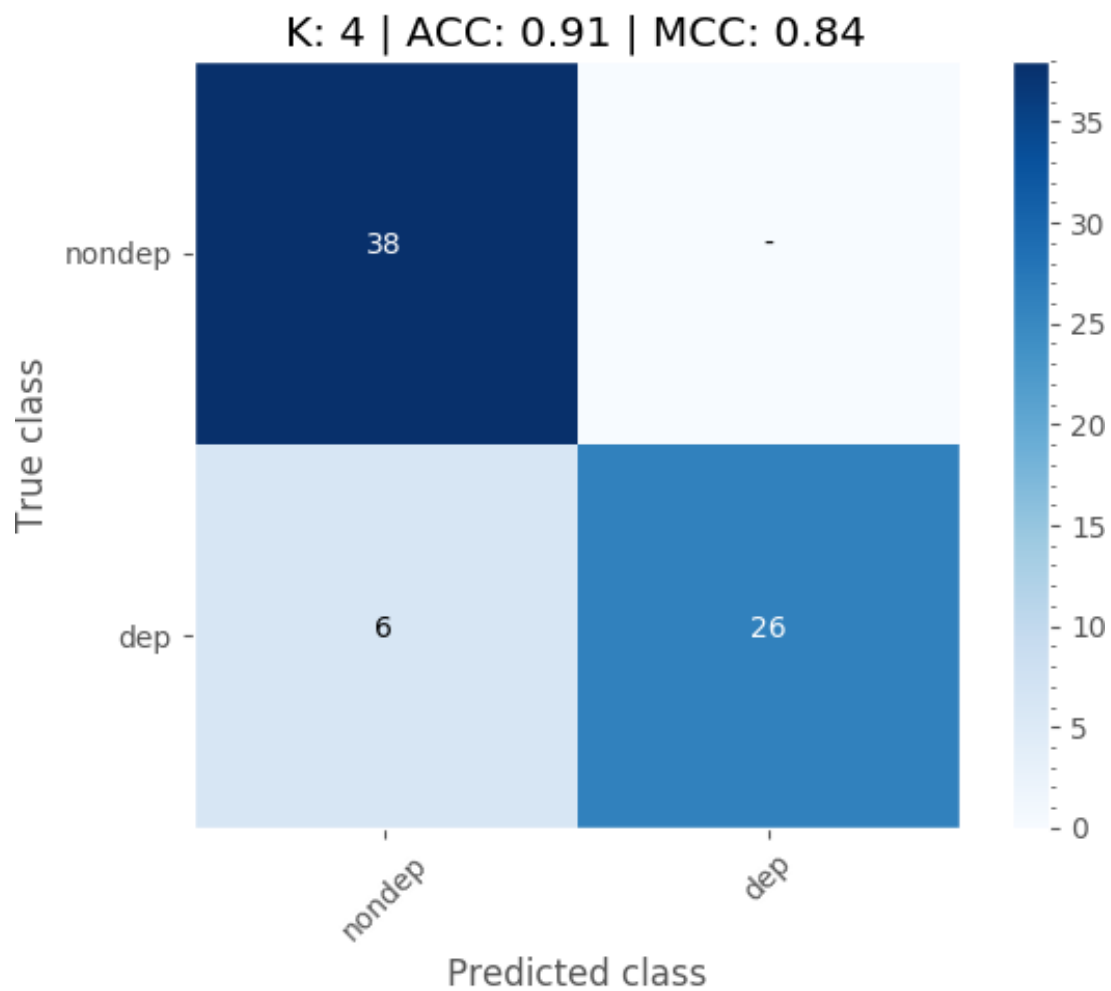
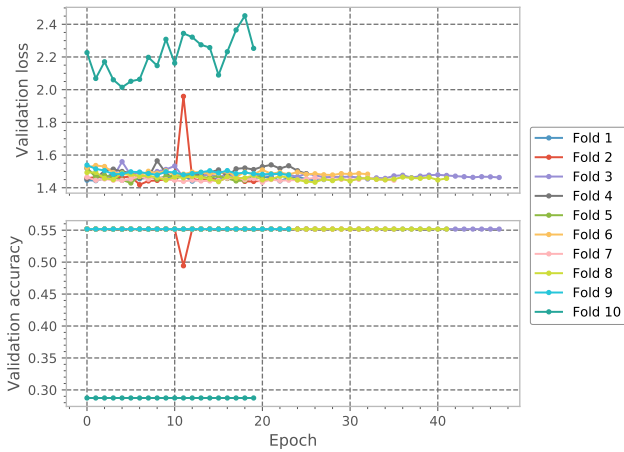
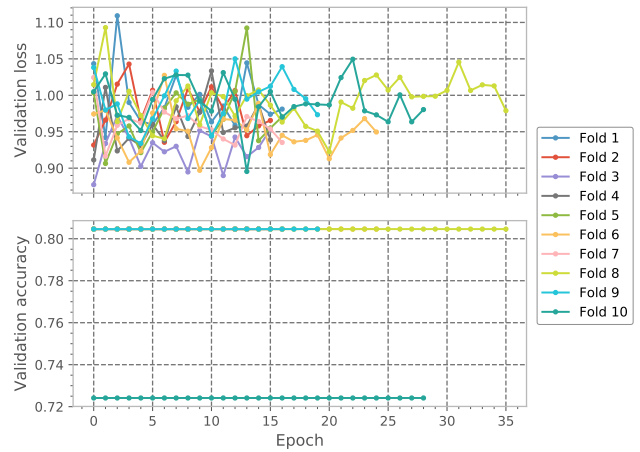


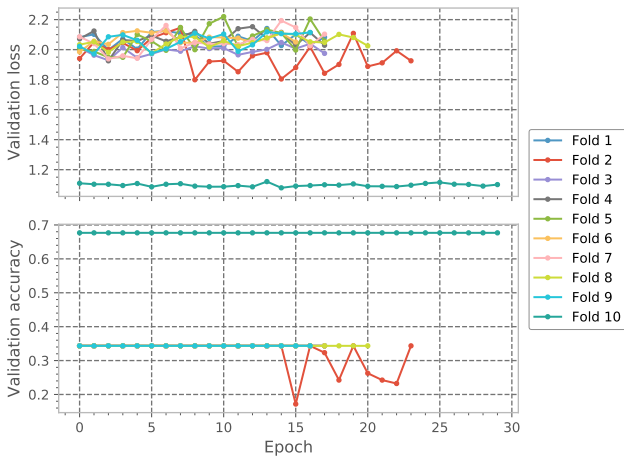
Figure B.2: The figure shows the confusion matrix for the best performing model that did not overfit. As described in the above appendix, the convergence for this particular fold is desirable, in which the model achieved an accuracy of 0.91 and MCC of 0.84.



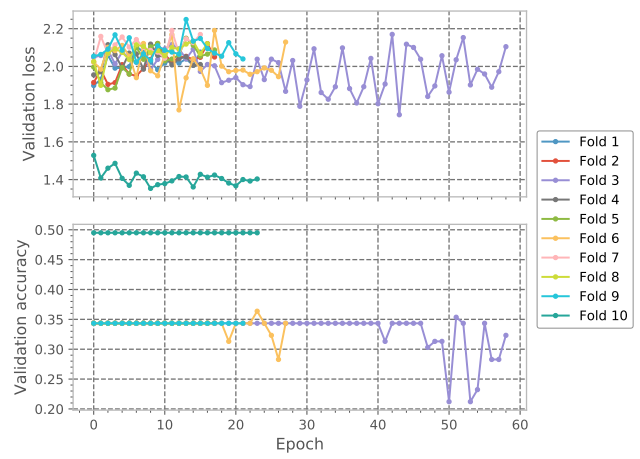
(a) 10 in batch size and sequence window of 30.
Momentum is 0.1



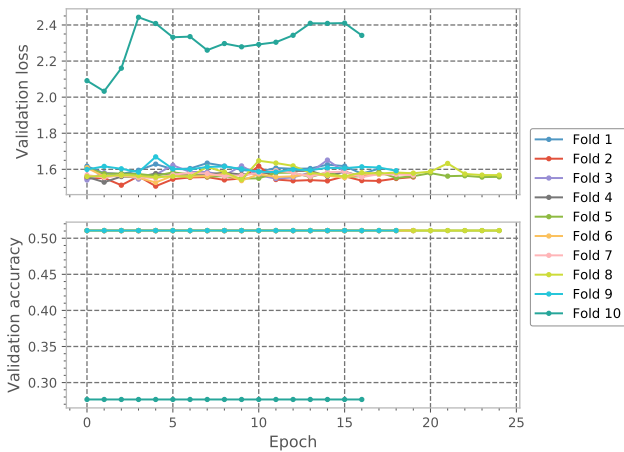
(b) 1 in batch size and sequence window of 30.
Momentum is 0.1.



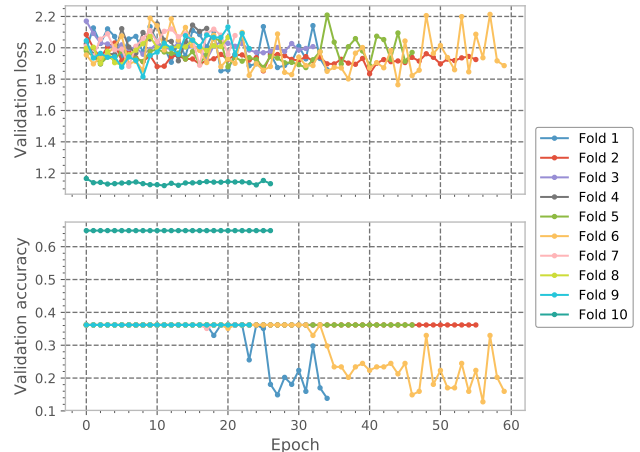
(c) 8 in batch size and sequence window of 14.
Momentum is 0.2.



(d) 10 in batch size and sequence window of 14.
Momentum is 0.2.

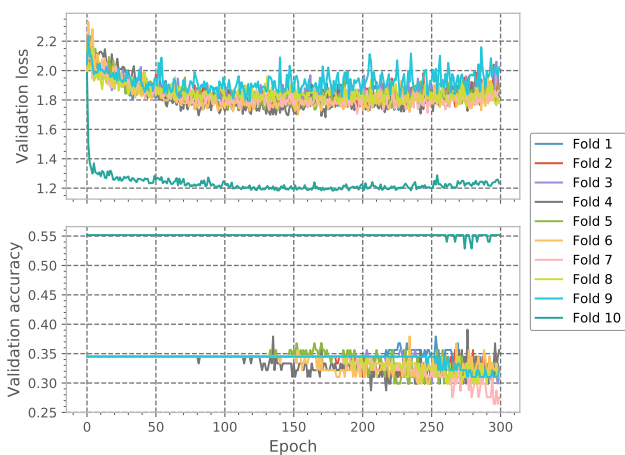


(e) 10 in batch size and sequence window of 21.
Momentum is 0.2 with Nesterov momentum.

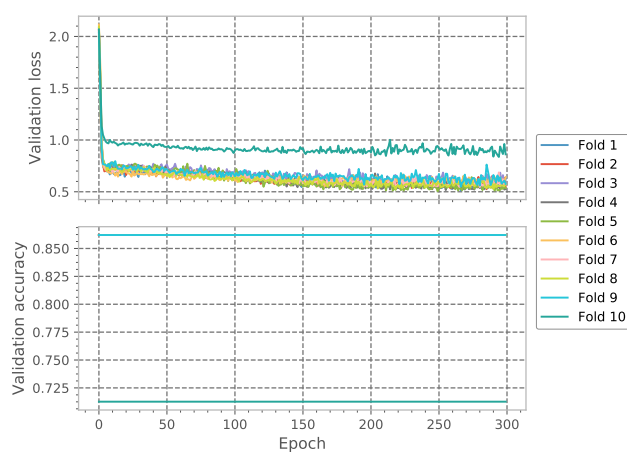


(f) 15 in batch size and sequence window of 21.
Momentum is 0.1.

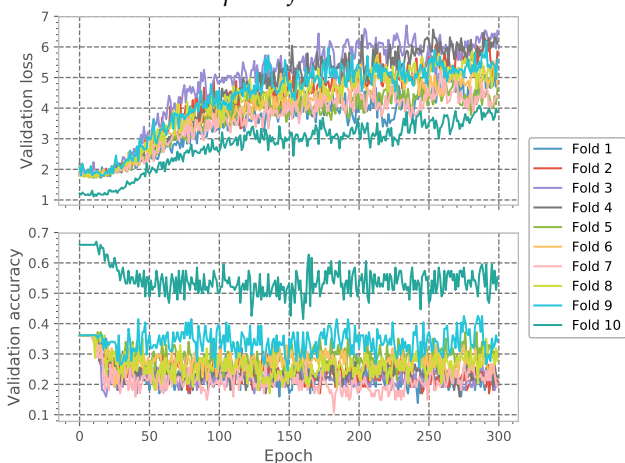
Figure B.3: The figures show the validation history for 6 different LSTM-results in the EV use case. All experiments used a learning rate of 0.001, Nadam as optimiser, dropout rate of 0.4 and ReLU as activation function.



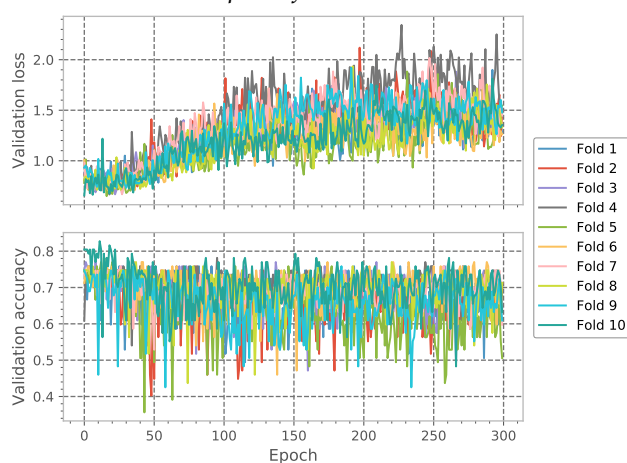
(a) Batch size of 5, sequence window of 30, dilation rate of 1 and kernel size of 5. SGD is used with a dropout of 0.2 and ReLU activation.



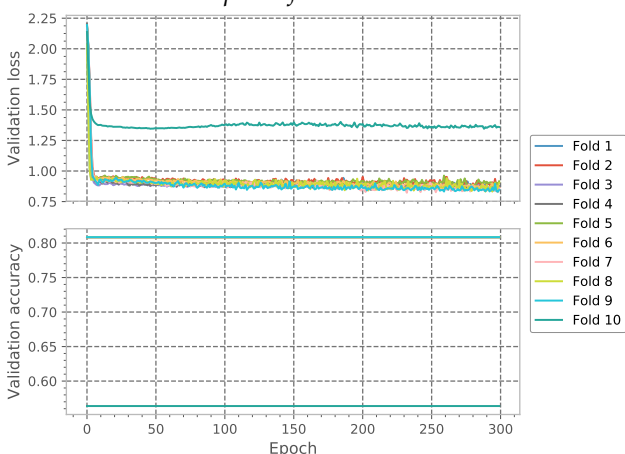
(b) Batch size of 5, sequence window of 30, dilation rate of 3 and kernel size of 8. SGD is used with a dropout of 0.2 and ReLU activation.



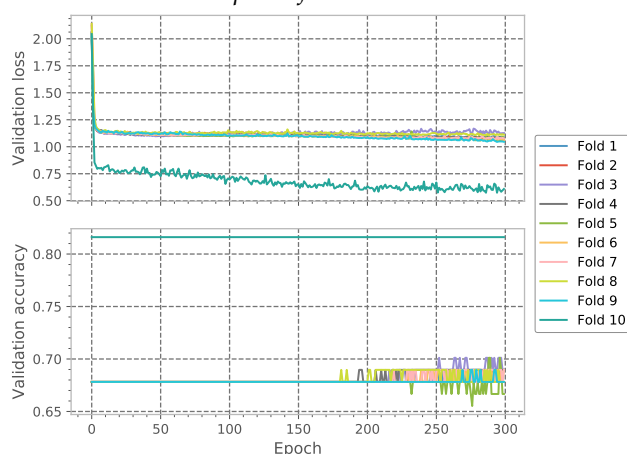
(c) Batch size of 5, sequence window of 21, dilation rate of 2 and kernel size of 4. Nadam is used with a dropout of 0.3 and ReLU activation.



(d) Batch size of 1, sequence window of 30, dilation rate of 2 and kernel size of 3. Nadam is used with a dropout of 0.3 and Tanh activation.

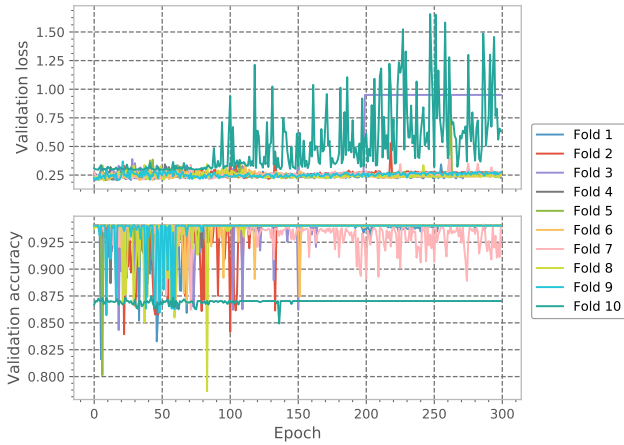


(e) Batch size of 10, sequence window of 21, dilation rate of 1 and kernel size of 3. SGD is used with a dropout of 0.4 and ReLU activation.

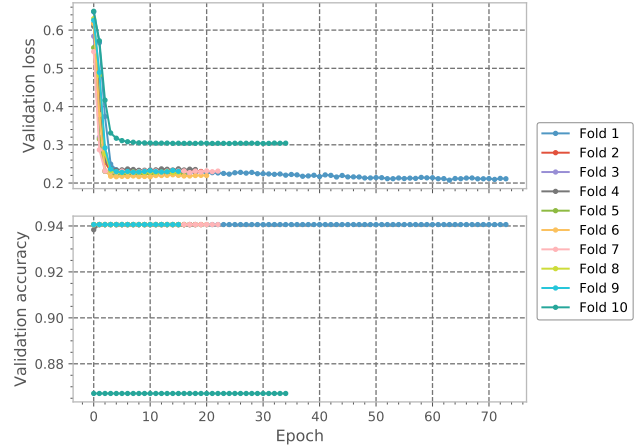


(f) Batch size of 5, sequence window of 30, dilation rate of 3 and kernel size of 5. SGD is used with a dropout of 0.2 and ReLU activation.

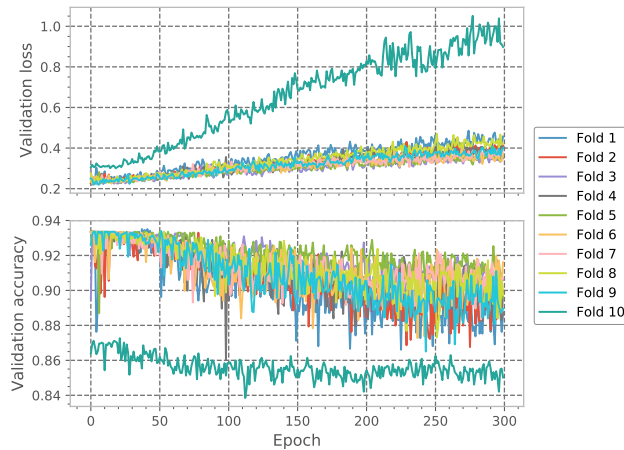
Figure B.4: The figures show the training history for 6 different initial CNN-results in the EV use case, where multiple configurations of hyperparameters were combined through trial and error. In many occasions, the loss convergence was ideal, although the classification task and accuracy convergence could be improved. All experiments used a momentum of 0.3 and learning rate of 0.001.



(a) RMSprop-optimiser and sequence window of 30. Momentum of 0.4, kernel size of 4 and dilation rate of 3.



(b) SGD-optimiser and momentum at 0.2. Sequence window of 35, kernel size of 4 and dilation rate set to 3.



(c) Nadam-optimiser, momentum of 0.4, kernel size of 3, dilation rate of 1 and sequence window of 14.

Figure B.5: The figures show the validation history for three different CNN-experiments in the football use case. While the results were not comparable to those presented in the thesis, the figures show the training progress on various hyperparameter configurations. All experiments used a learning rate of 0.001 and a batch size set to 15.

Metric (sd) Experiment	ACC	F1	MCC	PREC	REC	SPEC
Depression CNN R_1	0.820 (0.117)	0.819 (0.137)	0.345 (0.236)	0.867 (0.088)	0.820 (0.117)	0.635 (0.162)
Depression CNN R_2	0.780 (0.202)	0.785 (0.219)	0.542 (0.364)	0.874 (0.116)	0.780 (0.202)	0.813 (0.226)
Depression CNN R_3	0.816 (0.145)	0.811 (0.145)	0.411 (0.312)	0.844 (0.122)	0.816 (0.145)	0.629 (0.334)
Depression CNN R_4	0.801 (0.145)	0.791 (0.187)	0.422 (0.366)	0.813 (0.209)	0.801 (0.145)	0.711 (0.288)
Depression LSTM R_1	0.820 (0.167)	0.806 (0.175)	0.486 (0.424)	0.814 (0.172)	0.820 (0.167)	0.653 (0.302)
Depression LSTM R_2	0.788 (0.178)	0.792 (0.185)	0.594 (0.258)	0.887 (0.089)	0.788 (0.178)	0.858 (0.152)
Depression LSTM R_3	0.818 (0.130)	0.823 (0.124)	0.504 (0.294)	0.858 (0.113)	0.818 (0.130)	0.789 (0.167)
Energy CNN R_1	0.600 (0.141)	0.484 (0.178)	0.039 (0.039)	0.428 (0.191)	0.600 (0.141)	0.425 (0.149)
Energy CNN R_2	0.601 (0.133)	0.489 (0.143)	0.038 (0.048)	0.449 (0.135)	0.601 (0.133)	0.426 (0.157)
Energy LSTM R_1	0.582 (0.200)	0.466 (0.211)	-0.012 (0.026)	0.399 (0.199)	0.582 (0.200)	0.410 (0.190)
Football CNN R_1	0.892 (0.033)	0.946 (0.015)	0.222 (0.245)	0.909 (0.024)	0.993 (0.008)	0.139 (0.055)
Football CNN R_2	0.887 (0.038)	0.945 (0.016)	0.147 (0.249)	0.907 (0.024)	0.994 (0.010)	0.134 (0.055)
Football LSTM R_1	0.899 (0.028)	0.947 (0.016)	0.283 (0.218)	0.915 (0.020)	0.985 (0.017)	0.152 (0.059)
Football LSTM R_2	0.900 (0.030)	0.948 (0.017)	0.277 (0.203)	0.915 (0.022)	0.986 (0.014)	0.155 (0.057)

Table B.1: *The table shows a compact summary of all experiments, with the average metric evaluations and corresponding standard deviations in parantheses.*

Appendix C

Methodology

Appendices related to the methodology chapter.

C.1 Example of model building in Keras

A generic example of how to build a model in Keras by stacking layers.

```
1 from keras.layers.core import Dense, Dropout
2 from keras.layers.recurrent import LSTM
3 from keras.models import Sequential
4
5 # Model specifications
6 model = Sequential()
7 classes = 10
8 layers = [150, 75, 50, 25]
9
10 # Stack layers with units
11 for num_units in layers:
12     model.add(LSTM(num_units))
13     model.add(Dropout(0.2))
14
15 # Add dense layer and compile model
16 model.add(Dense(classes, activation='softmax'))
17 model.compile(loss='categorical_crossentropy', optimizer='sgd')
18
19 # Model fitting and prediction, assuming X and y is defined
20 model.fit(X, y)
21 model.predict(X_test)
```

Listing C.1: The listing showcases an example of how to build a simple model in Keras by stacking 4 hidden LSTM-units with dropout in each layer and lastly a dense layer for optimisation, similar to our models in this the thesis.

C.2 Example of Pandas data manipulation operations

A generic example of various data manipulation operations in Pandas.

```
1 import pandas as pd
2
3 # Read Pandas dataframe, example on 1 patient from depression dataset
4 df = pd.read_csv('control_1.csv', index_col='timestamp', parse_dates=True)
5
6 # Resampling series: hourly average
7 df = df.resample('H').mean()
8
9 # Filling missing values: eg. with average
10 df = df.fillna(df.mean())
11
12 # Filling missing values: eg. with 24-hour rolling average
13 rolling_average = df.rolling(window=24).mean()
14 df = df.fillna(rolling_average)
15
16 # Visualising dataframes
17 df.plot()
18
19 # Grouping series: by hourly activity
20 hourly_activity = df.groupby(by=df.index.hour)
21
22 # Descriptive statistics: on hourly activity
23 hourly_activity.describe()
```

Listing C.2: *The listing shows an example of how Pandas can be used for time-series. Details regarding different operations like resampling and grouping is intuitively abstracted away in Pandas. By default, this example assumes there is only one column in the dataset, although in the case of multiple variables/columns, it is possible to specify which to perform an operation on.*

C.3 Configuration file used in thesis

The appendix shows the content of the configuration file used in the thesis, where we set model architecture, use case and various other configurations before running the experiment- and evaluation pipeline.

C.4 Execution script used in thesis

The execution script used to load configurations, build models and run the experiment- and evaluation pipeline.

```

1 {
2   "_usecase": "football",
3   "_experiment_folder": "exp_football",
4   "data": {
5     "batches": 5,
6     "classes": 4,
7     "epoch": 300,
8     "es_loss": true,
9     "es_patience": 10,
10    "folds": 10,
11    "future_window": 1,
12    "resample": "D",
13    "score_threshold": 3,
14    "test_size": 0.1,
15    "val_size": 0.1,
16    "window": 30
17  },
18  "network": {
19    "activation": "relu",
20    "activation_rec": "tanh",
21    "architecture": "cnn",
22    "dilation_rate": 3,
23    "dropout": 0.3,
24    "kernel_size": 5,
25    "layers": [
26      150,
27      75,
28      50,
29      25
30    ],
31    "learning_rate": 0.001,
32    "momentum": 0.001,
33    "nesterov": false,
34    "optimiser": "sgd",
35    "padding": "causal",
36    "sigmoid_threshold": 0.5,
37    "strides": 1
38  }
39 }

```

Listing C.3: Overview of configuration file used in thesis. It is a JSON-file, storing configurations in key/value pairs and is separated into data and network, for data-related and network-related parameters, mostly for convenience.

```

:
13 import usecases.energy as energy
14 import usecases.depression as depression
15 import usecases.football as football

:

104 if __name__ == '__main__':
105     comments = input('===_Comments_===\n>_')
106     plotting.set_styles()
107     config = Config(comments)
108     config.read_config()
109     config.validate()
110     data = None
111
112     if config.usecase == config.DEPRESSION:
113         dp = depression.PredictDepression(data_path='../data/depression')
114         data = dp.read_data(config.data['window'], config.data['resample'])
115     elif config.usecase == config.FOOTBALL:
116         fp = football.FootballPrediction(data_path='../data/football')
117         data = fp.read_data(config.data['window'])
118     else:
119         ep = energy.EnergyPrediction(data_path='../data/energy')
120         data = ep.read_data(config.data['window'])
121
122     print('|_Running_{}_...'.format(config.network['architecture']))
123     main(config, data)
124     print('|', 40 * '-')

```

Listing C.4: *Subset of run-script used in thesis to illustrate a general level overview on loading configurations and running experiments.*