LUDWIG-MAXIMILIANS-UNIVERSITY MUNICH
Department of Statistics

Master's Thesis

Influence of Preprocessing on Deep Learning Models

Joshua Wagner

Completion period:	24.09.2020 - 24.03.2021
Supervisors:	Dr. Roman Hornung
	Maximilian Mandl

Abstract

Deep learning models continue to gain popularity in various domains due to the increase in available data. The models are usually compared on their prediction performance after extensive hyperparameter tuning. An often unrecognised factor for the increase or decrease in performance of these models is the preprocessing of the data. This thesis proposes an extensive comparison study of various different preprocessing methods and their influence on model performance on the example of time-series data. Two time-series datasets are analysed with varying preprocessing methods and their influence on classification and forecasting performance of 12 different models. The experimental results for classification indicate a low variability of convolutional neural networks through preprocessing while maintaining high classification accuracy. The results also indicate that tree based machine learning models are more robust to data preprocessing variations than support vector or nearest neighbour classifiers. The addition of bi-directionality to a recurrent neural network has also been observed to decrease the variability due to preprocessing while improving performance. The forecasting experiments confirm the stability of CNNs with regard to preprocessing.

Contents

List of Figures

List of Tables

1 Introduction

The first step in most statistical analyses and applications is the preprocessing of data. This is necessitated by the fact that "raw" datasets are often not directly usable. This preprocessing includes many different choices a researcher can make. These choices can influence the performance of a deep learning model. For example faulty or noisy training data can impact the classification accuracy of Convolutional Neural Networks(CNN) (Yim and Sohn, 2017). Generally they include data cleaning, normalization, transformation, feature extraction and selection and data-type specific preprocessing steps (Kotsiantis et al., 2006). These methods can have a distinct impact on the generalization of a deep or machine learning model.

The first step, data cleaning, can vary from dataset to dataset. Some generally used steps are the treatment of missing values and the cleaning of noise in the data. There exist a multitude of algorithms for missing value treatment which can be loosely categorized into two fields: Maximum Likelihood Imputation and Machine Learning Based Methods (García et al., 2015). A method to mitigate noise can be instance selection which attempts to maintain or improve the generalization performance of the models while shrinking the sample size. A number of algorithms can be chosen for instance selection and typically depend on the selected model for the analysis and the dataset itself (Grochowski and Jankowski (2004) and Jankowski and Grochowski (2004)). A further cleaning step that reduces noise in the data is outlier detection which has a number of algorithms to choose from (Aggarwal and Yu (2001) and Hodge and Austin (2004)). These first and the following preprocessing steps are applicable to most data types but can vary depending on the data. Feature extraction can depend more on previous knowledge about the domain of the data than insights from generally applicable algorithms. Feature selection on the other hand does have some more generally applicable algorithms from which a researcher can choose (Saeys et al. (2007) and Chandrashekar and Sahin (2014)). These few general preprocessing steps already come with many researcher degrees of freedom since most steps have many different algorithms and hyperparameters to choose from as could be seen in the previous paragraph. The freedom of choice persists for other data domains where deep learning models are frequently used, e.g. for images, text or time-series.

Deep learning models like convolutional neural networks are commonly used for image classification. As with other data types, the images need preprocessing. While scaling of images is done to minimize the computational cost, other steps are again

optional. Two common further preprocessing steps for images are noise reduction in the training data and transformation of training data. Both methods are used to increase the performance of the model. Noise reduction produces sharper, clearer images which are easier to categorize for the model (Yim and Sohn, 2017), while transformation improves the ability of the model to generalize on new data (Tabik et al., 2017). These preprocessing steps come with different researcher degrees of freedom as many algorithms exist for both methods. For example centering, elastic deformation, translation, rotation, a combination of these, etc. can be chosen for the transformation of images.

The common choice for text analysis or Natural Language Processing (NLP) are deep learning models. These models cannot directly use text which necessitates the preprocessing into a number based representation. A traditional choice for this preprocessing were count-based distributional models. These are more and more replaced by neural network based word embedding models. The embedding models themselves contain many hyperparameters, some predefined or implicit, others can be freely tuned to specific problems. These hyperparameters can account for much of the performance gains of word embedding algorithms and consequently for the subsequent analysis model (Levy et al., 2015). Other preprocessing steps, e.g. lemmatizing, lower-casing etc., can further impact the model performance and are generally under-represented steps in publications (Camacho-Collados and Pilehvar, 2017).

A wide variety of preprocessing exist for time-series analysis which in turn enable models to analyse the data in various ways. Common choices for preprocessing would be deseasonalization of seasonal trends in the data or a log-transformation of the data (Ahmed et al., 2010). Another option would be the sliding window size for the transformation of a continuous time series into a discrete series (Naduvil-Vadukootu et al., 2017). Other approaches like multiresolution analysis and denoising are also usable for time-series data and further introduced in this thesis. Data after these preprocessing steps retains the form of a time-series and can be analysed with various different models. For example, the analysis could use a recurrent neural network with different cells, a convolutional neural network adapted to 1-dimensional data or other machine learning models. A different preprocessing approach can be used for classification of less common time-series data, e.g. human activity data with acceleration signals. The preprocessing involves a transformation into an image which can be done with a number of different algorithms. These include raw plots of the time-series, a multichannel image in which x-,y- and z-coordinates of the acceleration data are taken as colour channel

data, a spectrogram through short-time Fourier transformations or a scaleogram with a continuous wavelet-transform (Zheng et al. (2018) and Madine et al. (2019)). These are just some preprocessing steps which enable different models for time-series prediction and classification. This thesis further focuses on the impact of preprocessing methods for deep learning models on the example of time-series data.

The next chapter contains an overview of other publications which study the influence of preprocessing on deep learning models. The third chapter explains the methods used in this thesis while the fourth chapter introduces the datasets which were used for the experiments. The fifth chapter contains the results and lastly a conclusion is drawn in the sixth chapter.

2 Related Publications

In comparison to the vast number of publications about deep learning models and their applications, few investigate the influence of pre-processing on the performance. Publications that do often examine only one type of data which can be attributed to the differences in preprocessing that exist for the different data types that deep learning models can handle, e.g. images, text and time-series data.

Chandrashekar and Sahin (2014) examine the influence of feature selection on machine and deep learning models on the example of the classification performance of a Support Vector Machine (SVM) and a Radial Basis Function Network (RBFN) with seven time-series datasets and two different selection methods. The results show that feature selection methods almost always improve the performance of the models.
Ahmed et al. (2010) compare eight different machine and deep learning models on their prediction performance for time-series with three differently preprocessed input variables. They do not investigate the influence of the preprocessing to the entire dataset as it was out of scope for their publication. The results show that the influence of preprocessing on the performance varies by preprocessing and model.
Liu et al. (2013) experiment with different models to predict wind speed. While they use Discrete Wavelet and Discrete Wavelet-Packet Decomposition to predict based on stable sub-series of the input time-series and show that decomposed models achieve better performance, they do not examine the influence of the decomposition depth or other preprocessing methods. The results show that models trained on the wavelet packet decomposition perform better than their counterparts without.
Naduvil-Vadukootu et al. (2017) compare different machine and deep learning models for regression and classification on time-series data and examine the influence that balancing and stratification of a dataset can have on the classification performance. They do not further analyse the influence of other preprocessing steps. They show that stratification and balancing of the data both have positive impact on the performance of the models. The performance increase varies for each model.
Zheng et al. (2018) examine the impact that a transformation from numerical time-series into an image can have on classification performance for Human Activity Recognition (HAR). They process the time-series into four different images: raw plots of the time-series, multichannel images where the colour channels correspond to the x-,y- and z-axis, a spectrogram which represents the frequency features of the signal as the magnitude squared of the short-time Fourier transform (STFT), and a combination of a spectrogram and shallow features. They also investigate the influence of the length of

segment into which a HAR time-series is preprocessed, but do not investigate further into other preprocessing methods. Their results show that the transformation of Human activity data into images can yield performance increases in comparison to machine learning models trained with features.

Evrendilek (2014) compare the prediction performance of three different deep learning models with four different preprocessing decisions: no denoising, denoising through discrete wavelet transformation(DWT) with either Coiflet, Daubechies and Symmlet wavelets families as bases. They do not examine other preprocessing methods. The results in the publication indicate that wavelet denoising with optimized parameters can increase the performance of deep learning models.

Schlüter and Deuschle (2010) examine the influence of wavelet based preprocessing on the performance of different models for time-series prediction. Their publication compares classical statistical models, like ARMA/ARIMA, with different wavelet based preprocessing steps. Their variation in preprocessing include three different wavelet families: Haar, Daubechies and Morlet. Their publication is limited to traditional statistical models and does not further examine the influence on newer models. Their results indicate that the performance of the models can be increased with wavelet based preprocessing. The decision if wavelet denoising or multiresolution analysis is the preferable preprocessing depends on characteristics of the data.

The additions of this thesis in respect to the above is the comparison of multiple different deep learning models with machine learning and traditional statistical models. This thesis also contributes the explicit analysis of the variability of the models due to preprocessing. The preprocessing variations in this thesis are also more varied as in the above mentioned publication although they do build upon them. The preprocessing variations include as first step denoising with 12 different choices: no denoising, median filter denoising and wavelet based denoising with five different wavelets and two different thresholds. The further preprocessing methods include scaleograms from continuous wavelet transformation with three different wavelets and two maximum scales, multiresolution analysis with six different variations and feature extraction with six variations. The models are also more extensive as in the above seen publications as they include twelve different models: five deep learning models, five machine learning and two traditional statistical models.

3 Methodology

Various different models and preprocessing methods are used in the analysis of the impact of preprocessing on the performance of Deep Learning models. Some of these models and preprocessing methods are used in domains outside of time-series data. Others are specialized for time-series analysis, e.g. an autoregressive integrated moving-average model. The next sub-sections therefore introduce the various different models and preprocessing methods that are used in the experiments.

3.1 Models

Multiple different deep learning models are used for time-series classification and regression. Additional models are used to compare the results and variability of the deep learning models for the classification and regression task. The additional models include two traditional statistical models: the multinomial regression for time-series classification and a seasonal autoregressive integrated moving-average model for the forecasting task which are introduced in Sub-Section 3.1.1. The various different machine learning models and their compositions are introduced in Sub-Section 3.1.2. Sub-Sections 3.1.3, 3.1.4 and 3.1.5 contain the architecture of the deep learning models. Sub-Section 3.1.6 contains two additional layers that are used in the deep learning models. Most of these models have multiple existing variations in the literature. The performance of deep learning models can also vary depending on their specific architecture. The following sub-sections therefore only include the architecture of the models that is used in this thesis. All hyperparameters are default values if not specifically declared in this thesis.

3.1.1 Traditional statistical models

The main focus of this thesis is the assessment of the influence of different preprocessing methods on various deep leaning models. Two traditional statistical models are used to generate a baseline measurement for the two different prediction tasks. Multinomial regression is chosen as the traditional model for the classification task and an autoregressive integrated moving-average model for the regression.

Multinomial Regression is used for categorical target variables. The multinomial regression is a direct generalization of the logistic regression to more than two categories. The probability for a category $Y_i \in Y_1, \ldots, Y_{K-1}$ is given as

$$P(Y_i|X;\beta) = \frac{\exp(\beta^{(i)}X)}{1+\sum_{j=1}^{K-1}\exp(\beta^{(j)}X)} \tag{1}$$

and

$$P(Y_K|X;\beta) = \frac{1}{1+\sum_{j=1}^{K-1}\exp(\beta^{(j)}X)} \tag{2}$$

where $\beta^{(i)}$ is the parameter vector corresponding to the $i-th$ target category, K the reference category and X the vector of covariates (Böhning, 1992). The multinomial regression in this thesis uses the version provided in the python library *scikit-learn* (Pedregosa et al., 2011) which additionally regularizes the model with $L2$. This changes the optimization problem of β from

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}}\ \ell(\beta) \tag{3}$$

to

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}}\ \ell(\beta) - \alpha_{l_2} R(\beta) \tag{4}$$

where $R(\beta) = ||\beta||_2^2$ is the sum of squared coefficients, α_{l_2} the parameter that specifies the strength of the regularization and $\ell(\cdot)$ is the multinomial log-likelihood (Ng, 2004).

ARIMA models are used for time-series data where past values of a variable are used to predict future values. The acronym stands for Auto-Regressive Integrated Moving-Average which are the parts that make up the model.
An auto-regressive model predicts future values y_t through

$$y_t = c + \varepsilon_t + \sum_{i=1}^{p} a_i y_{t-i}, \tag{5}$$

a combination of a constant c, an error term ε_t and p weighted past values $y_{t-p},\ldots,y_{t-1}$.
A moving-average model predicts future values y_t through a combination of a constant, an error term and a weighted moving-average of q past error terms with

$$y_t = c + \varepsilon_t + \sum_{j=1}^{q} b_j \varepsilon_{t-j} \tag{6}$$

where $\varepsilon_t \sim N(0,\sigma^2)$ and i.i.d..

A combination of both models is called an Auto-Regressive Moving-Average model. The ARMA model uses the condition that the time-series is stationary, e.g. that the mean and variance of the time-series do not vary over time (Boshnakov, 2011). A continuous up- or down-ward trend in data can be rectified through differencing the series, e.g. $Z_t = Y_t - Y_{t-1}$ or $Z_t = (1-L)Y_t$ where L is the lag operator. A difference operation in an ARMA model is denoted through the addition of the parameter d which indicates the level of differencing. The resulting ARIMA model is written as $arima(p,d,q)$.

Real-world data can also include seasonality, e.g. monthly time-series data has a seasonal periodicity of $s = 12$ months. ARIMA models that also include additional seasonality correction are written as $arima(p,d,q)(P,D,Q)s$. P,D and Q are variations of the lower-case parameters but additionally use information about the seasonal periodicity s. $D = 1$ indicates a seasonal difference, e.g. $Z_t = Y_t - Y_{t-12}$ for monthly data with $s = 12$. $P = 1$ means that Y_t is lagged once, as with p, but with seasonal periodicity, e.g. Y_{t-12} for $s = 12$. Q is likewise the seasonal counter-part to q and indicates the seasonal-lag of the error term components (Dagum, 1980, p. 8). The correct values for the parameters can be found through various different tests. This can be time consuming for the forecasting of a large number of time-series. Therefore automatic estimation of these parameters through the auto-arima function of the python package *pmdarima* (Smith et al., 17) is used. This methodology is also used in the baseline computation in the M4 competition (Makridakis et al., 2020) whose dataset is used for the forecasting comparison.

3.1.2 Machine learning models

Machine learning models use a data driven approach. Machine learning models, in contrast to the before seen traditional statistical models, thus come with less strict assumptions about the data and can generalize better. All machine learning models in this thesis use the implementation from the python library *scikit-learn* (Pedregosa et al., 2011). One such model is the k-nearest neighbours classifier which is introduced in the next paragraph.

K-Neighbours Classifier is a machine learning model that does not train in the conventional sense (Keller et al., 1985). Labelled training data is used to classify new data to one of the categories existing in the training data. This is achieved by majority voting with the k nearest data points of the training data to the new data point (Cover and Hart, 1967). The computation of distance to a data point is dependent on the distance metric used. The k nearest neighbour classifiers used in this thesis all use the Chebyshev distance defined as $D_{Chebyshev}(x_{train}, x_{new}) = \max_i(|x_{train_i}, x_{new_i}|)$ where x_{train} and x_{new} are data points from the training data or a new dataset respectively and where i denotes the feature i of x with which the distance is calculated. The choice of the number of neighbours k can also influence the performance of the model.

Decision Trees are as the name implies a tree-like structure of decisions which split the data into two parts at each step. The splitting points are commonly called nodes. At each node the data is split along one of the features of x which maximises a given

metric. Common choices for the metric are the information gain and the Gini-impurity. The decision trees used in this thesis all use the Gini-impurity which is defined for a node as $G(p) = 1 - \sum_i^J p_i^2$ where p_i is the fraction of data points in the node with class i of the J classes (Coppersmith et al., 1999). Trees with this metric further split the data until either a pre-defined maximum tree depth is reached or the data in a node consists only of one class.

Random Forest models are an ensemble of multiple decision trees. The decision trees use bootstrapped samples of the training data for each tree. Each tree predicts a class for a given input x. The class which was predicted by the most trees is chosen as output (Breiman, 2001). This methodology counteracts the over-fitting that can occur when only a single decision tree is used.

Support Vector Machines separate data with two classes with a maximal margin between the decision boundary and the data. The support vectors are composed of data points that lie within the margin. The separation of data is sometimes easier to accomplish in a different dimensionality. Therefore a projection ϕ of the data x into a different dimensionality is used. For data vectors $x_i \in \mathbb{R}^p, i = 1,\ldots,n$ and $y \in \{1,-1\}^n$ the support vector machine tries to find parameters $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that $y_i(w^\top \phi(x_i) + b) = 1$ for most data. While allowing some data points to be miss-classified one can write this as a primal problem (Bishop, 2006, pg. 332)

$$\begin{aligned} \min_{w,b,\zeta} \frac{1}{2} w^\top w + C_{dist} \sum_{i=1}^{n} \zeta_i \\ \text{subject to } y_i(w^\top \phi(x_i) + b) \geq 1 - \zeta_i, \\ \zeta_i \geq 0, i = 1,\ldots,n \end{aligned}$$

where ζ_i is the distance of a miss-classified data point from the correct margin boundary and C_{dist} is the strength with which these distances are penalized. This is called a *soft margin* as it allows data points to be miss-classified as is the case when the distribution of the two classes overlap. Primal problems are often rewritten as dual problems as they are easier solved through quadratic programming. The dual problem is given as

$$\begin{aligned} \min_{\alpha_i} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{subject to } 0 \leq \alpha_i \leq C_{dist} \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}$$

where $k(x_i, x_j)$ is a kernel that projects the data into a higher dimensional space and defined as $k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$ (Bishop, 2006, pg. 333). The kernel used in this thesis is the radial basis function and defined as $\exp(-\nu\|x - x'\|^2)$ where $\nu = \frac{1}{\#\text{features} \cdot \sigma^2(x)}$ (Pedregosa et al., 2011). For a more detailed introduction to support vector machines and their application in regression see Bishop (2006, pg. 326-345). The above described two-class classification can be extended to multiple classes through the training of multiple two-class SVMs for each combination of classes as is the case in the implementation in the *scikit-learn* library.

Gradient Boosting Trees are additive models which predict y_i for an input x_i with $\hat{y}_i = F_M(x_i) = \sum_{m=1}^{M} h_m(x_i)$ where h_m is a decision tree of a fixed size and M is the number of boosting steps to take. Gradient boosting trees are computed iteratively where the next ensemble of trees is calculated by $F_m(x) = F_{m-1}(x) + h_m(x)$. The new tree h_m is fitted to minimize a sum of costs

$$h_m = \underset{h}{argmin}\, C_m = \underset{h}{argmin} \sum_{i=1}^{n} C(y_i, F_{m-1}(x_i) + h_m(x_i)) \tag{7}$$

where $C(y_i, F_{m-1}(x_i))$ denotes a cost function. This is done by fitting the weak learner h_m to the pseudo-residuals

$$r_{im} = -\left[\frac{\partial C(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \tag{8}$$

for $i = 1, \ldots, n$ which is the negative gradient evaluated at $F(x) = F_{m-1}(x)$ (Friedman, 2001). The update can make large leaps in the direction of the negative gradient for large gradient values. This is called an exploding gradient which leads updates to overshoot the optimal value of $\hat{y}_i$. A step size ρ can be calculated to regulate this problem for each boosting step m. The computation of a step-size ρ_m for the boosting approach is in some cases non-trivial. The optimization of a step-size and a weak learner without gradients in $\{\beta_m, a_m\} = \underset{\beta, a}{argmin} \sum_{i=1}^{N} C(y_i, F_{m-1}(x_i) + \beta h(x_i; a)$ is difficult and thus replaced by the gradient approach (Friedman, 2001). It can be obtained for any differentiable loss through line-search when it holds that

$$a_m = \underset{a, \beta}{argmin} \sum_{i=1}^{N} [r_{im} - \beta h(x_i; a)]^2 \tag{9}$$

with

$$\rho_m = \underset{\rho}{argmin} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i; a_m). \tag{10}$$

This approach replaces the optimization with regard to the gradient with the closely correlated constrained $h(x; a_m)$. This results in the updated prediction $F_m(x) = F_{m-1}(x) +$

$\rho_m h(x;a_m)$ (Friedman, 2001). The optimization in the direction of the negative gradient is called *gradient descent* and is further introduced in the next sub-section.

3.1.3 Neural Network

The following sub-sections all contain models that are commonly known as deep learning models. They differ in their architecture but are all based on the neural network model introduced in this sub-section. Deep learning models vary from the machine learning and statistical models in the previous sub-sections in that they are fitted multiple times on the same dataset. The models are adjusted with each fit to improve their predictive capabilities.

The easiest neural network(NN) is a so-called multilayer perceptron. It consists of at least three layers: the input, output and one or multiple hidden layers. The case with one hidden layer is considered in the following. Each layer is made out of the so-called neurons. Each neuron contains a non-linear activation function. Various different non-linear functions can be chosen for this activation function. A historically common choice is the sigmoid function defined as $\sigma(x) = \frac{1}{1+\exp(-x)}$. For a vectorized notation the sigmoid $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ is defined for a vector $z \in \mathbb{R}^m$ in a component-wise manner $(\sigma(z))_i = \sigma(z_i)$ (Higham and Higham, 2019). Each neuron also contains a bias b_i which is added to the weighted input combination of the outputs from the previous layer. Therefore the output of neuron i is defined as

$$\sigma(\sum_j w_{ij} a_j + b_i) \tag{11}$$

where a_j is the output of neuron j from the previous layer weighted with w_{ij} and neuron i specific bias b_i. The vectorized notation for a layer is then given as

$$\sigma(Wa + b) \tag{12}$$

where W is the weight matrix, a the input vector and b the bias vector. As each layer takes the output of the previous layer as input we can define the output of a three layer multilayer perceptron as

$$F(x) = \sigma(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}) \tag{13}$$

where $W^{[3]}, W^{[2]}$ are the weight matrices of layer 2 and 3 respectively and $b^{[2]}, b^{[3]}$ are the corresponding bias vectors. The first layer is the so-called input layer and simply returns the input x to every neuron in the next layer. How well the output $F(x)$ predicts the actual target variable y can be measured through a *cost* or *loss function* as in Sub-Section 3.1.2.

A quadratic cost function for an input of size N can be defined as

$$\text{Cost}(F(x)) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2. \tag{14}$$

The goal of minimizing the cost involves the optimization of the weights and biases of the model. The optimization of all weights and biases is done through the *gradient descent* method mentioned in Sub-Section 3.1.2. The gradient of the cost function is given as $\nabla\text{Cost}(F(x))$ where the derivative with respect to the r-th parameter of $F(x)$ is given by

$$(\nabla\text{Cost}(F(x)))_r = \frac{\partial\text{Cost}(F(x))}{\partial F(x)_r}. \tag{15}$$

Let p denote the vector of parameters of $F(x)$ which should be optimized. The updated parameters denoted by $p+\Delta p$ with a small perturbation Δp decrease the cost function when $\Delta p = -\nabla\text{Cost}(p)$. A step-size η similar to ρ_m in Sub-Section 3.1.2 is chosen to limit the update of p to small values and results in $p \to p - \eta\nabla\text{Cost}(p)$ (Higham and Higham, 2019). The calculation of $Cost(p)$ for the entirety of x_N can be computationally infeasible for dataset where N is large. The update is therefore approximated through a small sample $\{x^{\{k_i\}}\}_{i=1}^{m}$ of size $m << N$ from the entire dataset with (Higham and Higham, 2019)

$$p \to p - \eta\frac{1}{m}\sum_{i=1}^{m}\nabla C_{x^{\{k_i\}}}(p). \tag{16}$$

$\frac{N}{m}$ updates are called one *epoch* and represent one training cycle over the entire dataset. Deep learning models are usually trained over multiple epochs. The samples can be chosen with replacement or K samples of size m can be chosen without replacement if $N = Km$. This is called *stochastic gradient descent* as the gradient of the entire dataset is approximated through the sample. The parameter η which is also called the *learning rate* of the model is commonly replaced by more advanced schemas which vary the learning rate throughout the training, e.g. AdaGrad (Duchi et al., 2011), Adam (Kingma and Ba, 2017), etc..

The specific calculation of the partial derivatives of the gradient is done through a method called *back propagation*. For a neural network with L layers the cost function can be written as $C = \frac{1}{2}\|y - a^{[L]}\|_2^2$ where $a^{[L]}$ is the output of the layer L. The output $a^{[l]}$ for $l = 2,\ldots,L$ is the non-linear transformation of the weighted input $z^{[l]}$ and is defined as $a^{[l]} = \sigma(z^{[l]})$. The weighted input $z^{[l]}$ is defined as $z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$ where $a^{[l-1]}$ is the output of the previous layer. The vector of partial derivatives $\delta^{[l]}$ with respect to the j-th neuron at layer l is defined by

$$\delta^{[l]} = \frac{\partial C}{\partial z_j^{[l]}} \tag{17}$$

for $1 \leq j \leq n_l$ and $2 \leq l \leq L$ where n_l is the number of neurons at layer l. The component-wise multiplication of two vectors, denoted by $\odot$, in the following definitions is defined by $(x \odot y)_i = x_i y_i$. The computation of the partial derivatives of hidden layers is possible through the application of the chain rule and is defined as:

$$\begin{aligned}
\delta^{[L]} &= \sigma'(z^{[L]}) \odot (a^L - y), \\
\delta^{[l]} &= \sigma'(z^{[l]}) \odot (W^{[l+1]})^\top \delta^{[l+1]} && \text{for } 2 \leq l \leq L-1, \\
\frac{\partial C}{\partial b_j^{[l]}} &= \delta_j^{[l]} && \text{for } 2 \leq l \leq L, \\
\frac{\partial C}{\partial w_{jk}^{[l]}} &= \delta_j^{[l]} a_k^{[l-1]} && \text{for } 2 \leq l \leq L.
\end{aligned}$$

where $1 \leq j \leq n_l$, $1 \leq k \leq n_{l-1}$ and $\sigma'()$ is the derivative of the non-linear activation function σ. For proof of the definitions above see Higham and Higham (2019, pg. 11-13). The definitions in this sub-section describe the so-called dense layer used in a multilayer perceptron. The name of the dense layer is given by the fact that every single neuron is connected to every neuron of the previous layer which can result in a large number of weights. The following sub-sections introduce further deep learning models that employ more complicated specialized layers in addition to dense layers.

3.1.4 Convolutional neural networks

A convolutional neural network(CNN) (Le Cun et al., 1989) is a specialized neural network. It uses the specialized convolutional and pooling layers in addition to the dense layer seen in Sub-Section 3.1.3. This allows for better performance of the model on data types like images.

The structure of convolutional layers can be described with a number of filters, also called kernels. For a 2-dimensional input a filter is convolved over the entire input and produces an output which is often referred to as feature map. The convolution for a 2-dimensional input of size 3×3 and a filter with size 2×2 can be defined as

$$\begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \tag{18}$$

where $*$ is the convolution operation, I denotes the input, w denotes the weights of the filter and the matrix s is the output. The results s_{ij} of the convolution are calculated as

weighted sum of parts of the input where

$$\begin{aligned} s_{11} &= I_{11} \cdot w_{11} + I_{12} \cdot w_{12} + I_{21} \cdot w_{21} + I_{22} \cdot w_{22}, \\ s_{12} &= I_{12} \cdot w_{11} + I_{13} \cdot w_{12} + I_{22} \cdot w_{21} + I_{23} \cdot w_{22}, \\ s_{21} &= I_{21} \cdot w_{11} + I_{22} \cdot w_{12} + I_{31} \cdot w_{21} + I_{32} \cdot w_{22}, \\ s_{22} &= I_{22} \cdot w_{11} + I_{23} \cdot w_{12} + I_{32} \cdot w_{21} + I_{33} \cdot w_{22}. \end{aligned} \tag{19}$$

A non-linear activation function as seen in Sub-Section 3.1.3 is then applied to each output s_{ij}. The output of a convolution is smaller than the input if no padding is used as only so-called valid convolutions are performed which need an input for each weight. Padding *pads* the edges of an input with a chosen value and allows the centre of the filter to overlap the outermost values of the input. It also allows the output of a filter to keep the dimensionality of the input and therefore allows for repeated application of convolutional layers (Yamashita et al., 2018). The Equation 18 uses a stride size of 1 which is the distance between two applications of the kernel. Larger stride sizes allow for downsampling of the feature maps. As can be seen in Equation 19 the weights of the filter are shared between convolutions with different parts of the input. This reduces the number of weights that have to be stored. Convolution is therefore more efficient than the dense matrix multiplication used in dense layers in regards to the memory requirements (Goodfellow et al., 2016). A third dimension of an input, e.g. different colour channels of an image, changes the filters to three dimensional objects. The calculation is similar to Equation 19 and produces again output values s_{ij} regardless of the depth of the input. The convolutional layer therefore collapses the third dimension of the input. Multiple filters are usually used and the outputs stacked along the last dimension. The convolutional layer therefore replaces the depth of the last dimension with the number of filters used.

The second layer that differentiates convolutional neural networks from standard neural network is the *pooling* layer. A pooling layer also has filter size, stride and padding as parameters but does not have any learnable parameters (Yamashita et al., 2018). Max pooling (Zhou and Chellappa, 1988) for example returns simply the largest value in the filter. A max pooling layer would compute the output s_{11} as $s_{11} = \max(\{I_{11}, I_{12}, I_{13}, I_{21}, I_{22}\})$. Common choices for stride and filter size are such that the repeated applications of the filter do not overlap(Yamashita et al., 2018). Different from convolution, pooling preserves the depth of the input feature map.

Convolutional neural networks can be applied to any grid-like data structure. This also applies for sequential data as they can be seen as a 1-dimensional grid (Goodfellow et al.,

2016). A specialized neural network for sequential data is the recurrent neural network which is introduced in the next sub-section.

3.1.5 Recurrent Neural Networks

Analysing sequential data like signals or time-series requires an influence of past values on future vales. A neural network architecture which is specialized for these temporal connections is the recurrent neural network(RNN). A recurrent network uses a hidden internal state $\tilde{h}$ that is updated with each new input x_t at time-step t. It can generally be defined as $\tilde{h}^{(t)} = f(\tilde{h}^{(t-1)}, x^{(t)}; \theta)$ where f is some transformation of the input $x^{(t)}$ with the past value $\tilde{h}^{(t-1)}$ and some additional parameters θ. The parameters grouped under θ include weights W for past hidden states, a bias term b and weights U for the input. An update to the hidden state can then be defined following the definition of a layer in Sub-Section 3.1.3 with $\tilde{h}^{(t)} = \sigma(b + Wh^{(t-1)} + Ux^{(t)})$ although the sigmoid function σ is usually replaced by tanh as non-linear activation function. Tanh is a sigmoidal function defined between -1 and 1 (Goodfellow et al., 2016). An unfolded recurrent neural network that makes a prediction only at the end of the input series can be seen in Figure 3.1.

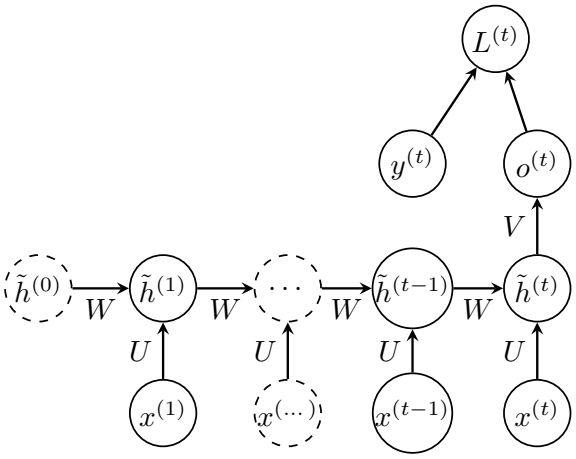

Figure 3.1: A time-unfolded RNN which predicts only once the entire input sequence is processed. Such models are used for time-series classification or so-called single-shot predicition. o denotes a dense layer which transforms the last internal state into an output.

Such a model is called a single-shot prediction model as it predicts the entire output only once at the end of the input sequence. Other types of recurrent neural networks and more information about the specific computations such as back propagation through time can be found in Goodfellow et al. (2016) or Graves (2012).

A simple extension of a recurrent network is the bi-directional recurrent neural network. This model uses a second recurrent layer which analyses the input in the reverse temporal direction. This allows for long-term dependencies in the data to be easier captured than through a normal recurrent neural network. A recurrent neural network struggles with long-term dependencies as is shown in Bengio et al. (1994). Some types of data and recurrent network architecture do not allow for the use of a bi-directional layer. For example financial data with a recurrent network which predicts at every time-step t could not efficiently use bi-directionality. The use of different hidden units allows for the modelling of long-term dependencies without bi-directionality in the layer.

A popular replacement for the hidden unit is the long-short term-memory unit denoted by LSTM in the following. A LSTM unit replaces the relatively simple recurrent unit with a more complicated update schema composed of a forget gate $f_i^{(t)}$, an input gate $i^{(t)}$, a cell state $s^{(t)}$ and a output gate $q_i(t)$ in addition to the previous hidden state $\tilde{h}^{(t-1)}$ as can be seen in Figure 3.2. The forget gate is defined as

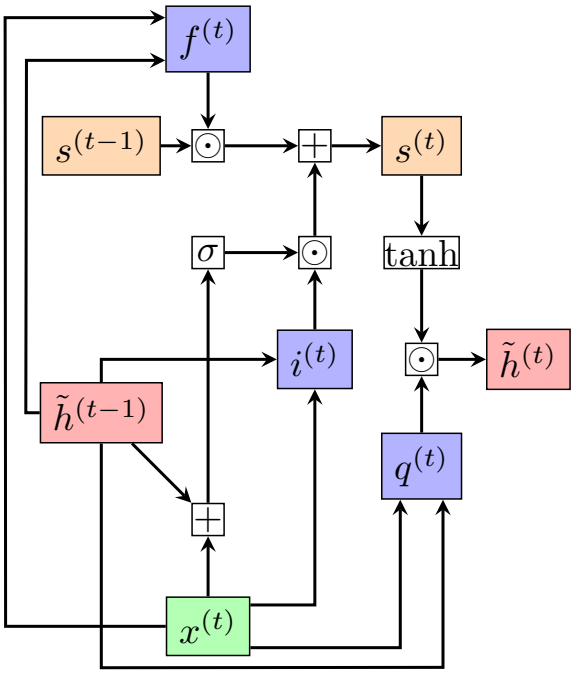

Figure 3.2: A LSTM unit visualized with its components. Hidden states are marked in red, input in green, gates in blue, internal states in orange and operations are transparent.

$$f^{(t)} = \sigma(b_f + U_f x^{(t)} + W_f \tilde{h}^{(t-1)}) \tag{20}$$

where b_f, U_f, W_f are respectively biases, input and recurrent weights of the forget gate. The internal state $s^{(t)}$ is updated with

$$s^{(t)} = f^{(t)} \odot s^{(t-1)} + i^{(t)} \odot \sigma(b + U x^{(t)} + W \tilde{h}^{(t-1)}) \tag{21}$$

where $\odot$ denotes the element-wise multiplication and b, U, W are the biases, input and recurrent weights respectively. The computation of the input gate $i^{(t)}$ is done in the same manner as the forget gate but with its own bias and weights b_i, U_i, W_i as

$$i^{(t)} = \sigma(b_i + U_i x^{(t)} + W_i \tilde{h}^{(t-1)}). \tag{22}$$

The output of the LSTM cell is calculated as

$$\tilde{h}^{(t)} = q^{(t)} \odot \tanh(s^{(t)}) \tag{23}$$

where the output gate $q^{(t)}$ is composed as (Goodfellow et al., 2016, pg. 406)

$$q^{(t)} = \sigma(b_q + U_q x^{(t)} + W_q \tilde{h}^{(t-1)}). \tag{24}$$

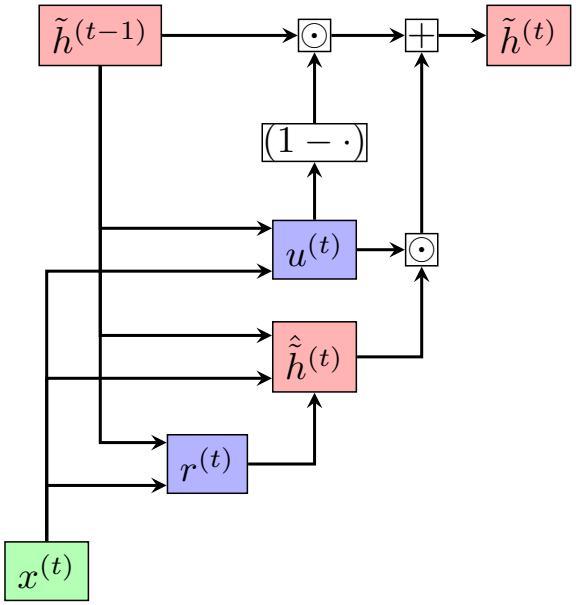

Figure 3.3: GRU circuit: hidden states in red, input in green, gates in blue and operations are transparent.

An alternative to the LSTM is a gated recurrent unit or GRU. It is composed of a reset gate $r^{(t)}$ and an update gate $u^{(t)}$. The update gate simultaneously controls the forgetting factor and the decision to update the hidden state as can be seen in Figure 3.3. A new hidden state in a GRU is calculated as

$$\tilde{h}^{(t)} = (1 - u^{(t)}) \odot \tilde{h}^{(t-1)} + u^{(t)} \odot \hat{\tilde{h}}^{(t)} \tag{25}$$

where

$$u^{(t)} = \sigma(b_u + U_u x^{(t)} + W_u \tilde{h}^{(t-1)}) \tag{26}$$

is the update gate and

$$\hat{\tilde{h}}^{(t)} = \tanh(b + Ux^{(t)} + W(r^{(t)} \odot \tilde{h}^{(t-1)})) \tag{27}$$

is the temporary hidden state where

$$r^{(t)} = \sigma(b_r + W_r x^{(t)} + U_r \tilde{h}^{(t-1)}) \tag{28}$$

is the reset gate (Chung et al., 2014). The reset gate can reset the hidden state if it is close to 0 and effectively remove any influence from past values on the hidden state. The previous definition of a GRU can be done without biases as in Chung et al. (2014) or with biases as in Goodfellow et al. (2016).

A combination of the LSTM and CNN architecture as described in Karim et al. (2018) for time-series classification is also used in this thesis. The model gives the same input to a single layer recurrent neural network with LSTM units and to a 3-layered CNN model. The outputs of both models are then concatenated and used as input for a single dense layer which returns the predicted class. The model also uses two additional sub-layers: a dropout layer and batch normalization. The full model architecture can be seen in Figure A.1 of the appendix.

3.1.6 Additional layers

Deep neural networks can overfit on the training data which increases their generalization error. Multiple additions to a neural network can be made to counteract this. Two of these additions are the batch normalization and the dropout layers which are introduced in the following paragraphs.

Batch Normalization is a method to regularize the outputs of a layer. The distribution of each layers input changes over time with training as the underlying parameters of previous layers change (Ioffe and Szegedy, 2015). This slows training down as it requires careful initialization of parameters and low training rates. Ioffe and Szegedy (2015) introduce the batch normalization (BN) layer to resolve this. The batch normalization is a normalization of the input to the next layer during a minibatch stochastic gradient descent update. The batch normalization is formally defined for a value x_i of Batch B with m values $x_{1,\dots,m}$ as (Ioffe and Szegedy, 2015)

$$BN(x_i) = \kappa \cdot \left(\frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}\right) + \beta$$

where κ and β are optimizable parameters, ε is a small constant value for numerical stability, μ_B is the mean of the minibatch values and σ_B^2 is the variance of the minibatch values. This stabilizes and regularizes the inputs of the layers and allows for higher learning rates (Ioffe and Szegedy, 2015). For inference on validation and testing datasets the running-means and variances during training are stored and used for normalization during inference (Ioffe and Szegedy, 2015).

Dropout layers are used to counteract the problem of overfitting (Hinton et al., 2012). A dropout layer randomly sets inputs to the following layer to 0 with a choosable rate during training. The sum over all inputs is unchanged as the inputs not set to 0 are scaled by $\frac{1}{(1-\text{rate})}$. The publication by Hinton et al. (2012) introduces dropout with a frequency of 0.5 although much more aggressive dropout rates can be used, e.g. the LSTM-CNN mixed model from Karim et al. (2018) uses a dropout rate of 0.8 for its RNN component.

3.2 Preprocessing Methods

The models are trained with various different data preprocessing methods. The possible preprocessing methods vary depending on the task and can influence the performance of different models. The preprocessing methods also enable the use of different models, e.g. a two-dimensional convolutional neural network can be trained with scaleogram images from a continuous wavelet transform.

3.2.1 Feature Extraction

Some models can make use of temporal dependencies in time-series data. It is also a necessity for time-series forecasting which uses past observations of the target variable $y_t, y_{t-1}, \ldots$ for the prediction of future values $y_{t+1}, y_{t+2}, \ldots$. These dependencies can prove difficult for models and even unnecessary for classification tasks. Time-series classification tasks like Human Activity Recognition(HAR) instead commonly make use of feature extraction or mapping (Anguita et al. (2013), Yang et al. (2008) and Khan et al. (2010)). These feature mappings allow the efficient use of new models, e.g. the multinomial regression. Which feature mappings are used is an arbitrary decision made by the researcher. They are usually based on already existing literature in the specific research field. The features in use for classification of time-series in this thesis are based on either the time-series itself (Anguita et al., 2013) or on a discrete wavelet transformation(DWT) (Mörchen, 2003). The feature calculations use the python package *tsfresh* (Christ et al., 2018) or a combination of base python and the package *PyWavelets*

(Lee et al., 2019).

The features mappings used on the raw time series are the $mean(x)$, $median(x)$, standard deviation $std(x)$, variance $var(x)$, minimum $min(x)$, maximum $max(x)$, 0.25-Quantile $q_{0.25}(x)$, 0.75-Quantile, the number of peaks in an interval, the mean absolute change

$$meanAbsChange(x) = \frac{1}{n} \sum_{i=1,\dots,n-1} |x_{i+1} - x_i| \tag{29}$$

and the absolute energy

$$E = \sum_{i=1,\dots,n} x_i^2. \tag{30}$$

More complex features include the sample entropy

$$SampEn(x) = -log(\frac{A}{B}) \tag{31}$$

with A calculated as

$$A = d[x_{m+1}(i), x_{m+1}(j)] < r \tag{32}$$

and

$$B = d[x_m(i), x_m(j)] < r \tag{33}$$

where $m = 2$, $r = 0.2 \cdot std(x)$, $x_m(i) = \{x_i, x_{i+1}, \dots, x_{i+m-1}\}$ and $d[x_m(i), x_m(j)]$ denotes the Chebyshev distance

$$D_{Chebyshev(x_m, x_{m+1})} = max_i(|x_{m_i}, x_{m+1_i}|). \tag{34}$$

Another complex feature are the autoregressive coefficients φ_i of a series

$$x_t = \varphi_0 + \sum_{i=1}^{k} \varphi_i X_{t-i} + \varepsilon_t \tag{35}$$

with $k \in \{1, \dots, 10\}$ (Christ et al., 2018). The last two feature mappings in use are the Skew, the third standardized moment $\tilde{\mu}_3$, and the Kurtosis, the fourth standardized moment $\tilde{\mu}_4$, of the Fourier Transformation $\mathscr{F}$ (see the next Section 3.2.2) of the time-series x (Anguita et al., 2013).

Another variation of feature mapping uses the coefficient series from Discrete Wavelet Transformation (see Section 3.2.4) (Mörchen, 2003). The feature mappings for the coefficient series include various quantiles q_k with $k \in \{0.05, 0.25, 0.5, 0.75, 0.95\}$, the mean, standard deviation and variance of a coefficient series. They also include the number of zero and mean crossings (Eyben et al., 2010). Lastly the entropy as seen in the

previous paragraph was calculated for the coefficients (AlSharabi et al., 2016).

The Fourier and wavelet transformation both yield representations of the signal in the frequency domain. Features calculated on this representation of the signal yield information about the underlying frequencies of the model. This can be beneficial if different classes in classification tasks vary in their underlying frequencies.

3.2.2 Fourier Transformation

Feature extraction for time-series data can use various methods as seen in the previous Sub-Section. The standard features (Khan et al. (2010) and Yang et al. (2008)) for HAR only provide information about the time-series itself. Newer features as in Anguita et al. (2013) make use of the Fourier transformation to extract information about the frequency components of a time-series.

The Fourier transform has different variants in use. The original continuous transform is for a function of time $f(t)$ given as

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) \cdot \exp(-i2\pi t\omega) dt \tag{36}$$

for a frequency ω. The more commonly used *discrete* Fourier transform (DFT) for a discrete sequence of numbers $\{x_n\} = x_0, x_1, \ldots, x_{N-1}$ transformed into a complex sequence of numbers $\{X_k\} = X_0, X_1, \ldots, X_{N-1}$ can be defined as

$$X_k = \sum_{n=0}^{N-1} x_m \cdot \exp(-2\pi i \frac{kn}{N}) \tag{37}$$

which is also the definition used for the implementation in the python library *numpy* (Harris et al., 2020) which is used in this thesis. The discrete Fourier transformation is especially popular due to the development of the fast Fourier transform (FFT) by Cooley and Tukey (1965). It reduces the computational complexity of the DFT from $O(N^2)$ to $O(N\log(N))$ and makes it one of the most popular algorithms in signal processing. A drawback of the Fourier transformation is that frequencies are extracted at a global level as can be seen in Figure 3.4. This limits its use for signals with time varying frequencies. The Fourier transform in Figure 3.4 correctly extracts the frequencies of the two sinus waves that are part of the signal. The switch from $x(t) = \sin(2 \cdot \pi \cdot 10 * t)$ to $x(t) = \sin(2 \cdot \pi \cdot 50 * t)$ at $t = 0.5s$ in the signal can not be inferred which limits its use in real world applications.

A Fourier-related approach to extract changes of frequency over time is the Short-time Fourier transform (STFT). The STFT breaks a signal into equally sized windows

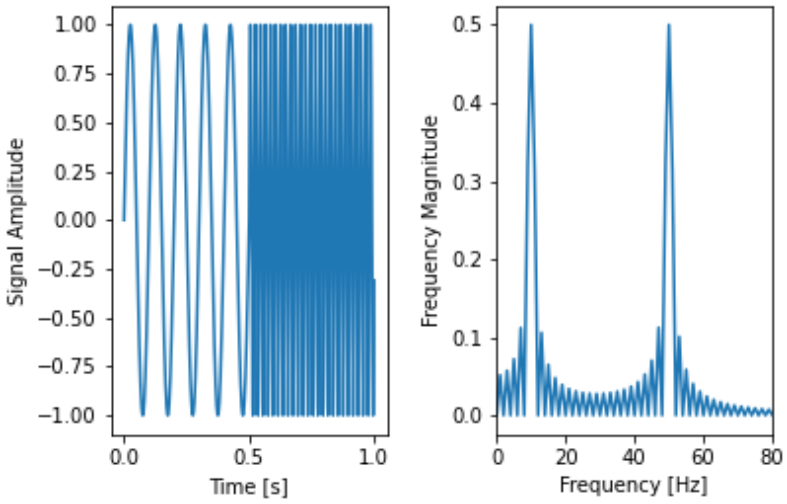

Figure 3.4: A signal with a 10 and 50 Hz sinus curve sampled at 1000 Hz, switching at 0.5s, and its corresponding FFT.

and returns the Fourier transform for each. The main drawback is the equal size of time windows which limits the result to having a high resolution in either time or frequency as illustrated in Figure 3.5.

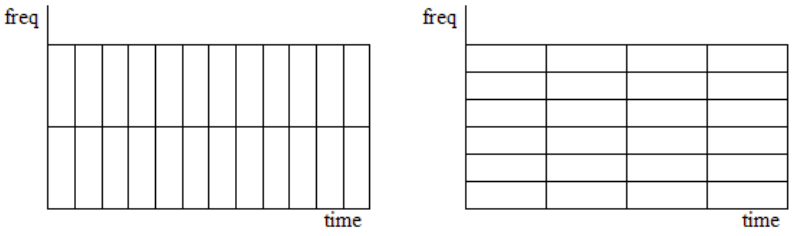

Figure 3.5: The frequency/time resolution trade-off of the STFT. Narrow window size as in the left part of the image result in good time resolution at the cost of poor frequency resolution. The inverse is true for the right part of the image.

Example Short-time Fourier transformations of the signal in the left part of Figure 3.4 can be seen in Figure 3.6. It can bee observed from Figure 3.6 that a small window size in time accurately depicts the change of the signal from 10 Hz to a 50 Hz at 0.5s. The trade-off between frequency and time resolution is also visible as the frequencies cannot be clearly inferred. The STFT with a window size of 1s extracts the frequencies more clearly as it is comparable to a standard Fourier transformation. The frequencies are accurately captured at 10 and 50 Hz although their position in the composite signal is

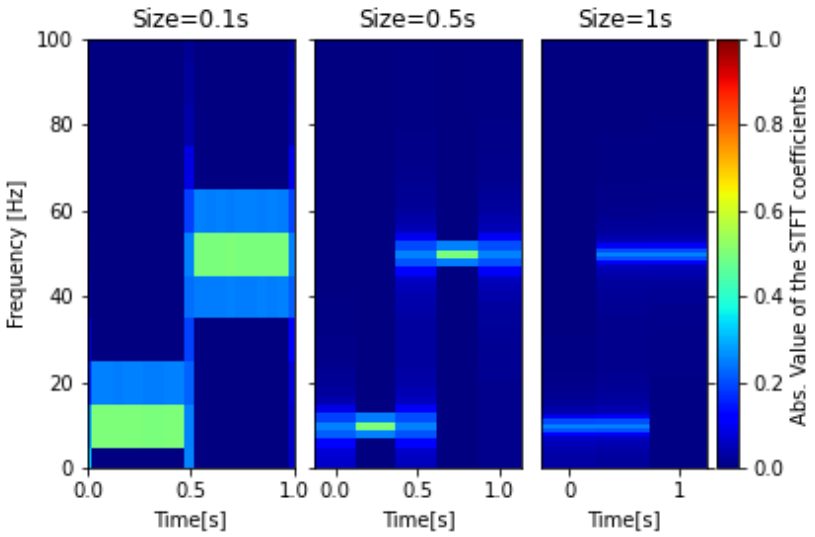

Figure 3.6: Spectrogramm of the short-time fourier transform of the signal seen in 3.4, with window sizes $\in [0.1, 0.5, 1]$ seconds. The uncertainty principle is visible as low window size has a high time resolution but bad frequency resolution while large window size has good frequency resolution but bad time resolution.

no longer as clearly visible as in the STFT with a small window size.

Transformations with variably sized windows allow for good resolution in both time and frequency domains. One such transformation is the wavelet transform which is introduced in the next sub-sections.

3.2.3 Continuous Wavelet Transformation

In context to the Fourier analysis the wavelet transformation is a relatively recent solution of the time- and frequency resolution trade-off. The wavelet transformation uses a fully scalable modulated window which is shifted along the signal and at each position the spectrum is calculated (Valens, 1999). The continuous wavelet transform(CWT) is formally written as

$$\gamma(s,\tau) = \int f(t)\psi^*_{s,\tau}(t)dt \tag{38}$$

where $*$ denotes the complex conjugation and $\gamma(s,\tau)$ are the wavelet coefficients. The function $f(t)$ is decomposed into a set of wavelets $\psi_{s,\tau}(t)$ with the new dimensions scale s and translation τ. The wavelets are a scaled and translated version of the mother wavelet ψ with

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\psi(\frac{t-\tau}{s}). \tag{39}$$

A wavelet has some base properties, namely the square of $\psi(\cdot)$ integrates to unity

$$\int_{-\infty}^{\infty} \psi^2(t)dt = 1 \tag{40}$$

and the average value of the wavelet in the time domain is zero

$$\int_{-\infty}^{\infty} \psi(t)dt = 0 \tag{41}$$

(Percival and Walden, 2000). The function $\psi(t)$ must therefore have a wave form. Wavelets must also satisfy the admissibility condition

$$\int \frac{|\Psi(w)|^2}{|w|} dw < +\infty \tag{42}$$

with $\Psi(w)$ being the Fourier transform of $\psi(t)$. This allows the reconstruction of a function from its continuous wavelet transform with the inverse wavelet transform

$$f(t) = \int\int \gamma(s,\tau)\psi_{s,\tau}(t)d\tau ds. \tag{43}$$

It also implies that the the Fourier transform of $\psi(t)$ vanishes at the zero frequency

$$\left|\Psi(w)\right|^2\Big|_{w=0} = 0. \tag{44}$$

The time-bandwidth product of a wavelet transform is the square of the input signal as can be seen from the previous paragraph which is not a desirable property for practical application. Therefore *regularity conditions* are imposed on the wavelet functions to make it decrease quickly with decreasing scale s (Valens, 1999). One such regularity condition for wavelets are the vanishing moments. The wavelet transform can be expanded into the Taylor series until order n with

$$\gamma(s,0) = \frac{1}{\sqrt{s}}[f(0)M_0 s + \frac{f^1(0)}{1!}M_1 s^2 + \frac{f^2(0)}{2!}M_2 s^3 + \cdots + \frac{f^n(0)}{n!}M_n s^{n+1} + O(s^{n+2})] \tag{45}$$

at $t = 0, \tau = 0$ with moments of a wavelet as

$$M_p = \int t^p \psi(t)dt \tag{46}$$

and f^p as the p^{th} derivative of f and $O(n+1)$ as the rest of the expansion.

The continuous wavelet transform is highly redundant through the continuous shifting of a continuously scalable function and therefore computationally expensive. The coefficients of the continuous wavelet transform can be used to generate a so-called scaleogram which is a 2-dimensional representation of the 1-dimensional signal. A method to alleviate this redundancy and reduce the computational cost is the use of discrete wavelets and their corresponding discrete wavelet transform which will be introduced in the next subsection. The different continuous wavelets used in this thesis can be found in Appendix A.2.

3.2.4 Discrete Wavelet Transformation

The discrete wavelets use a modified version of the representation of mother wavelets

$$\psi_{j,k}(t) = \frac{1}{s_0^j}\psi\left(\frac{t - k\tau_0 s_0^j}{s_0^j}\right) \tag{47}$$

where j and k are integers and $s_0 > 1$ is a fixed dilation step and τ_0 depends on this. Usual choices are $s_0 = 2$ and $\tau_0 = 1$ to achieve dyadic sampling of the frequency and time axis (Daubechies, 1992). The discrete wavelets are also made orthogonal to their own dilations and translations by choices of the mother wavelet with

$$\int \psi_{j,k}(t)\psi^*_{m,n}(t)dt = \begin{cases} 1 & \text{if } j = m \text{ and } k = n \\ 0 & \text{otherwise.} \end{cases} \tag{48}$$

A signal can then be reconstructed by summing the orthogonal wavelet basis functions $\psi_{j,k}$ weighted by the wavelet transform coefficients $\gamma(j,k)$ with (Valens, 1999)

$$f(t) = \sum_{j,k} \gamma(j,k)\psi_{j,k}(t). \tag{49}$$

Though the previous paragraph defines the discrete wavelets, the wavelet transformation itself is still continuous. The translation k is bounded by the length or duration of the signal. The scale parameter j though has no lower bound. As every stretch in the time domain by a factor of 2 halves its frequency bandwidth, one would need an infinite number of wavelets to cover the entire frequency spectrum (Valens, 1999). A solution in the form of a low-pass filter was introduced by Mallat (1989) with the so-called scaling function

$$\varphi(t) = \sum_{j,k} \gamma(j,k)\psi_{j,k}(t), \tag{50}$$

here decomposed into its wavelet components. This scaling function covers the entire spectrum otherwise covered by an infinite number of wavelets up to the scale j. The scaling function has an admissibility condition similar to the one seen in the previous sub-section with $\int \varphi(t)dt = 1$ which shows that the 0^{th} moment cannot vanish.

A signal can be analysed through an iterated filter bank which is done since the mid-1970's (Blu, 1993). This filter bank splits the signal with a high- and a low-pass filter. The same can be achieved with a wavelet transform as a high-pass and a scaling function as a low-pass filter at each scale j. The scaling function can then be split again with a high- and a low-pass filter for scale $j+1$. This is expressed in the multiresolution or two-scale relation formulation (Sidney Burrus et al., 1998) with

$$\varphi(2^j t) = \sum_k h_{j+1}(k)\varphi(2^{j+1}t - k) \tag{51}$$

where h_{j+1} is a weighting factor that forms a low-pass filter. Wavelets can then be expressed as translated scaling functions at the next scale with

$$\psi(2^j t) = \sum_k g_{j+1}(k)\varphi(2^{j+1}t - k) \tag{52}$$

where the weighting factors g_{j+1} form a high-pass filter. The original function $f(t)$ can then be expressed with dilated and translated scaling functions at scale j with

$$f(t) = \sum_k \lambda_j(k)\varphi(2^j t - k) \tag{53}$$

where λ_j are the scaling coefficients. Additional wavelets have to be added for scale $j-1$ in order to keep the same level of detail. $f(t)$ can then be written as

$$f(t) = \sum_k \lambda_{j-1}(k)\varphi(2^{j-1}t - k) + \sum_k \gamma_{j-1}(k)\psi(2^{j-1}t - k). \tag{54}$$

The coefficients $\lambda_{j-1}(k)$ and $\gamma_{j-1}(k)$ can be found with the inner products of $f(t)$ and $\varphi_{j,k}$ or $\psi_{j,k}$ respectively. Replacing $\varphi_{j,k}$ and $\psi_{j,k}$ with their representation through weighted scaling functions and some additional manipulation results in (Sidney Burrus et al., 1998)

$$\lambda_{j-1}(k) = \sum_m h(m-2k)\lambda_j(m); \tag{55}$$

$$\gamma_{j-1}(k) = \sum_m g(m-2k)\lambda_j(m). \tag{56}$$

This equation also defines the property of sub-sampling through the step-size of 2 in the variable k. Figure 3.7 shows this as one step of an iterated filter bank.

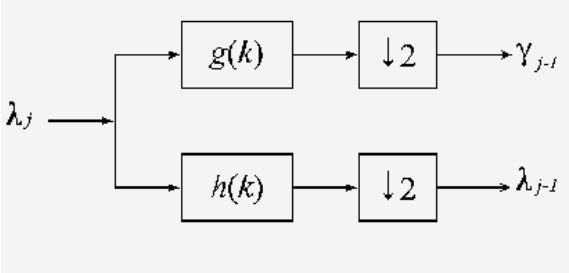

Figure 3.7: One step of an iterated filter bank from scale j to $j-1$. λ_j is split with scaling filter $h(k)$ and wavelet filter $g(k)$. Sub-sampling denoted through $\downarrow 2$. Fig. 5 from Valens (1999)

These coefficients are used in different preprocessing methods introduced in the next sub-section. The different discrete wavelets used in this thesis and examples of them can be found in Appendix A.3.

3.2.5 Discrete Wavelet Preprocessing

The previous sub-section shows how a signal $x(t)$ can be iteratively decomposed into scaling and wavelet coefficients λ and γ with scale j. The scaling and wavelet coefficients are also commonly called approximation cA and detail cD coefficients. The approximation generated with the low-pass filter $h(t)$ of the discrete wavelet transform contains the general lower-frequency trend of a signal. The detail coefficients contain the high-frequency components generated through the high-pass filter $g(t)$. These coefficients are generally reported with their respective decomposition level up to the maximal decomposition m for signal $x(t)$ as can be seen in Figure 3.8. These output coefficients of the discrete wavelet transform are used differently depending on the data preprocessing method used.

One such preprocessing method is the so-called multiresolution analysis of a signal. Multiresolution analysis predicts the coefficients $cA_m, cD_m, \ldots, cD_1$ of a discrete wavelet transform $DWT(\cdot)$ of the target time series $x(t+1), \ldots, x(t+p)$ directly based on the corresponding coefficients of the $DWT(\cdot)$ of the input time-series $x(t)$ where $x(t) = (x_1, \ldots, x_t)$. The coefficients of the target series are then inverted to yield a forecast (Conejo et al., 2005). The multiresolution analysis allows simultaneous prediction of high- and low-frequency parts of the signal through different models. An example would be a signal consisting of a low frequency trend and a high frequency oscillation which are both important for the prediction of the signal. This can be computationally expensive as each coefficient needs a new model. Renaud et al. (2005) only predict specific wavelet

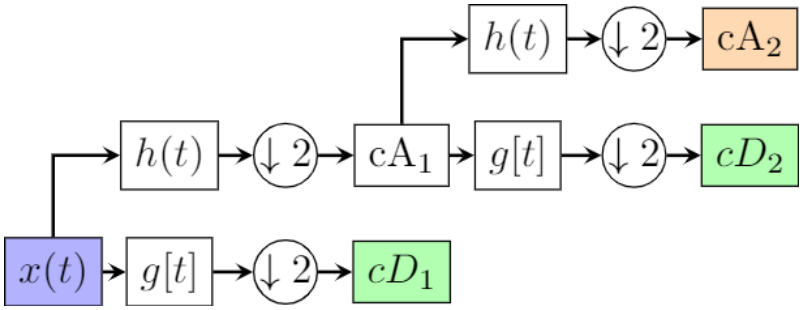

Figure 3.8: Multiresolution decomposition through DWT of a signal $x(t)$ in blue at decomposition level 2 with colour coded output coefficients. Generated output detail coefficients are marked green while the approximation coefficients are marked in orange. Sub- or down-sampling by a factor of 2 is marked with circles.

coefficients and invert with other coefficients set to zero. This is efficient as only one model has to be used but increases the forecasting error as parts of the original signal are lost. The multiresolution analysis that is used in this thesis uses all coefficients up to the chosen level based on Conejo et al. (2005).

Another preprocessing method based on the discrete wavelet transformation is the wavelet de-noising. It uses discrete wavelet decomposition to filter out high-frequency noise in a signal. This approach is based on the assumption that $x(t)$ is generated through a lower-frequency function $f(t)$ and additional white noise $\varepsilon(t) \sim N(0, \sigma^2), \sigma > 0$, e.g. $x(t) = f(t) + \varepsilon(t)$. The noise component is represented in each coefficient while the information about $f(t)$ is present only in a few coefficients (Nason et al., 2000). Therefore small coefficients only contain information about the noise in the signal. Consequently setting all coefficients below a threshold ι to zero and the following inversion of the modified wavelet coefficients yields a de-noised signal $\hat{x}(t)$. Different thresholding functions exist, though all de-noising in this thesis uses soft thresholding from Donoho and Johnstone (1994) defined as $WT' = sgn(WT)(|WT| - \iota)\mathbb{1}_{|WT|>\iota}$ where sgn denotes the signum function and WT the matrix of wavelet coefficients.

4 Datasets

Two datasets were used, one each for time-series classification and prediction. Both are publicly available at the websites of UCI and M4 respectively.

4.1 UCI-HAR dataset

The time-series classification experiment uses the Smartphone Based Human Activity Recognition(SBHAR) dataset from UCI. Version one of the dataset was released by Anguita et al. (2013) and contains tri-axial linear acceleration and angular velocity data. The data comes from a set of experiments with 30 volunteers wearing a hip-mounted smartphone. The participants freely performed a sequence of six different Activities of Daily Life(ADL), which were *walking*, *walking upstairs*, *walking downstairs*, *standing*, *sitting* and *laying down*. The experiment was done twice by each participant. The time-series data was filtered with a median filter and a 3rd order low-pass Butter-worth filter. The signals were then sampled in fixed-width sliding windows of 2.56sec and 50% overlap which corresponds, at an initial sampling rate of 50Hz, to signal length of 128 time-steps. The first version of the dataset only provides the preprocessed data and extracted features from the sliding windows. The classification comparisons in this thesis use version 2.1 which was released by Reyes-Ortiz et al. (2016). This new version contains the data from version 1.0 and expands it with new transitional labels *stand-to-sit*, *sit-to-stand*, *sit-to-lie*, *lie-to-sit*, *stand-to-lie* and *lie-to-stand*. The transitional labels only account for a small part of the dataset as they were not the main focus of the study and therefore less time was spent in transition.

4.2 M4 competition dataset

The M4 competition dataset (Makridakis et al., 2020) was initially compiled for the fourth competition in the M-Series. The dataset contains 100000 time-series split by their time interval of collection, e.g. weekly, monthly etc.. The dataset was generated by a random selection of these 100000 time-series from a database called ForeDeCk from the National Technical University of Athens that contains 900000 time-series(Makridakis et al., 2020). The database is built from diverse and publicly accessible data sources. It contains time-series from domains with a focus on business forecasting applications such as industries, services, tourism, imports and exports, demographics, education, labor and wage, government, households, bonds, stocks, insurances, loans, real estate, transportation, and natural resources and environment (Spiliotis et al., 2020). The time-series were anonymized and all features that could be used to identify the time-series were removed such as the time

frame when the series was collected. The dataset contains information from what domain the time-series were collected. The additional information is used to stratify the split into train, validation and test data. This allows the split data to each contain the same proportions of data from the different domains. The time-series prediction experiments in this thesis use the subset of monthly data for two reasons. The first is the volume of data available as the monthly subset contains with 48000 time-series nearly half of the entire M4 dataset. The second is that the monthly data is the most balanced subset in regard to their original domain with about 10000 time-series each from Micro, Industry, Macro and Finance domains. About 5700 time-series are from the Demographic domain and 277 are categorized as Other. The prediction task in this thesis uses the forecasting horizon from the M4 competition for monthly data which is a long-term forecasting horizon of 12 months.

5 Experimental Results

The models described in Section 3 are used to classify the time-series data from the human activity dataset and forecast the time-series from the M4 competition. The models are trained with various different preprocessing combinations which influence the performance of the models. The variability of the models over the preprocessings is measured as well as the average performance. Machine and deep learning models vary in their hyperparameters which are often tuned specifically to one dataset or task. These tuned hyperparameters can heavily influence the performance of models. As such the models in the experiments use the default hyperparameters wherever possible and the same seed for the initial weights for the deep learning models. Machine learning models with splitting or randomized feature selection like the decision tree use the same random seed for reproducibility. The deep learning models were each trained for 20 epochs and use the same architecture when possible, e.g. the CNN for images uses a two-dimensional convolutional layer while the CNN for time-series data uses an one-dimensional convolutional layer. The results therefore only apply to the models with the same hyperparameters as other choices could change the influence of the preprocessing.

5.1 Classification

The classification experiments use the Human Activity Recognition dataset introduced in Section 4. The raw acceleration and gyroscopic data is 3 dimensional and the acceleration data is further split into body and gravitational motion after denoising. Each of the 9 signals is then further preprocessed with one of three preprocessing methods: scaleogram from a continuous wavelet transformation, multiresolution analysis or feature extraction. Due to the overlapping nature of sequences the data is split based on the physical experiment from which they were obtained. This ensures that no data is shared between the training and test set. The first 41 experiments are used as training data, the next 10 experiments are validation data and the last 10 experiments are used as test data. The deep learning models evaluate the prediction accuracy during each epoch on the validation data. The weights of the model are saved to a file if the prediction accuracy on the validation data surpasses the previous epochs. This is a so-called best model checkpoint. This ensures that variability during training is mitigated as the deep learning models are trained for only 20 epochs. It also is a proxy measure for generalization and performance on the test set and minimizes the risk of overfitting on the training data. The machine learning models do not use the validation data as checkpoint and therefore are trained with an union of training and validation data.

The task of classifying time-series data enables non-reversible transformations. These transformations are in the following denoted as major preprocessings as they fundamentally change the data and enable different models, e.g. a 2-dimensional convolutional neural network for images created through CWT or feature extraction for multinomial regression. A minor preprocessing is the denoising of the time-series which has 12 variations. 10 of those are DWT thresholding (Sub-Section 3.2.5), 1 variation is the median filter which is also used in Anguita et al. (2013) and the last variation is the lack of any denoising. The wavelet denoising variations use two different thresholds $\iota \in \{0.2, 0.3\}$ and the wavelets families $\psi \in \{$Haar, Daubechies-4, D.-Mey., Symlets-4, Coiflets-4$\}$. The level of the DWT for denoising is set to the maximum possible level for each time-series.

The major preprocessing transformations enable different models for time-series classification. The analysis of a scaleogram as a result of continuous wavelet transform uses a convolutional neural network based on Zheng et al. (2018). The second major preprocessing uses multiresolution analysis. The baseline performance for models using time varying inputs is also analysed. For this the models are trained with the denoised time-series input. The multiresolution approach uses discrete wavelet transformation whose coefficients are then concatenated for ease of computation. The last major preprocessing is the feature extraction. This method allows a wider range of models to be employed that were not usable on the raw time-series data or the image representation.

The deep learning models that are used for classification include a 2-dimensional convolutional neural network(CNN) which uses two convolutional layer each followed by a max. pooling layer and finally two dense layers and an 1-dimensional CNN which uses the same architecture only with the corresponding 1-dim. layers. They also include a neural network(NN) composed of three dense layers, a recurrent neural network(RNN) which uses a single recurrent layer with gated recurrent units(GRU) followed by two dense layers, a second version of the RNN with bi-directionality in the recurrent layer and a recurrent-/convolutional neural network mixed model(LSTM-CNN) from Karim et al. (2018) whose architecture can be found in Figure A.1 in the appendix. The mixed model uses long-short term memory units (LSTM) in the recurrent part of the model followed by a dropout layer with a high dropout rate of 0.8. The convolutional part of the mixed model contains three convolutional blocks followed by a global average pooling layer. The convolutional blocks each contain a convolutional layer and a batch normaliza-

tion layer followed by an activation layer. The machine learning models include a support vector classifier(SVC), a decision tree classifier(DTC), a random forest classifier(RFC), a gradient boosting classifier(GBC) and a k-neighbours classifier(KNC). The statistical baseline model for classification is the multinomial regression(multinom. Reg.). A list of models per major preprocessing method can be found in Table 5.1.

Major Preprocessing	Models
CWT images	CNN
Multiresolution time	NN, RNN, CNN, Bi-RNN, LSTM-CNN
Feature extraction	NN, SVC, RFC, DTC, KNC, GBC, NN(sklearn), multinom. Reg.

Table 5.1: Table of the models and for which major preprocessing they were used.

The major preprocessings again come with their own variability. For example the choice of wavelet in the continuous or the discrete wavelet transformation. The addition of models trained with the original features from Reyes-Ortiz et al. (2016) increases the number of total models trained to 1016. The results are therefore not presented for each variation of a model but aggregated by their various preprocessings as can be seen in Figure 5.1, Figure 5.3 or Figure 5.2. The metric used to compare the models is the median classification accuracy on the test set and variability is measured through the interquartile distance(IQD) of the model performance over various preprocessings.

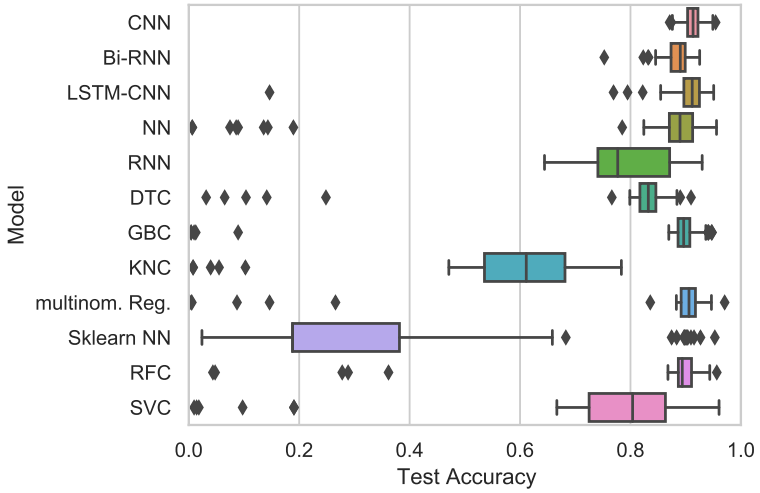

Figure 5.1: Model accuracy aggregated over all preprocessing variations.

As can be seen in Figure 5.1, the model performance varies strongly based on the

preprocessings. The outliers in the figure necessitate the use of robust aggregation measures such as the median instead of mean and interquartile distance instead of variance. The *sklearn* based neural network is the model with the greatest variability in accuracy. This is attributed to the wide performance range from epoch to epoch. The sklearn implementation does not allow for passing of validation data and only for the definition of the fraction of training data that is to be used as validation data. The overlapping nature of the sequences does not allow for this randomised splitting. Therefore the more robust neural network with a checkpoint using the Keras (Chollet et al., 2015) python library is also included for feature models.

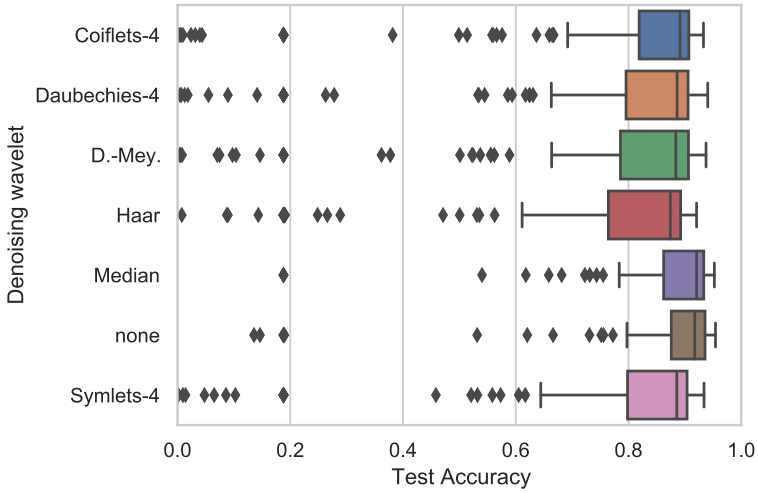

Figure 5.2: Aggregated accuracy of all models per denoising wavelet.

All major preprocessings use denoising as first step in the preprocessing pipeline. The original features use a median filter as mentioned in Section 4 but are excluded from the following analysis due to their lack of variability. From Figure 5.2 it can be seen that the median filter and no filtering have the highest median accuracy over all models and major preprocessings. The interquartile distance is also the lowest for these preprocessings as can be seen in Table B.3 in the appendix. This shows that the relatively easy median filter outperforms the shown wavelet denoising approaches for soft thresholds with 0.2 and 0.3. It is also observable that the median results only decrease from 0.920732 to 0.917429 accuracy if no denoising is done and even outperforms the wavelet denoising approaches. This indicates that bad denoising actually decreases performance for human activity recognition compared to no denoising.

From Figure 5.3 it can be seen that the models the with least variability use either the features from Reyes-Ortiz et al. (2016) or the scaleogram images from the CWT. A

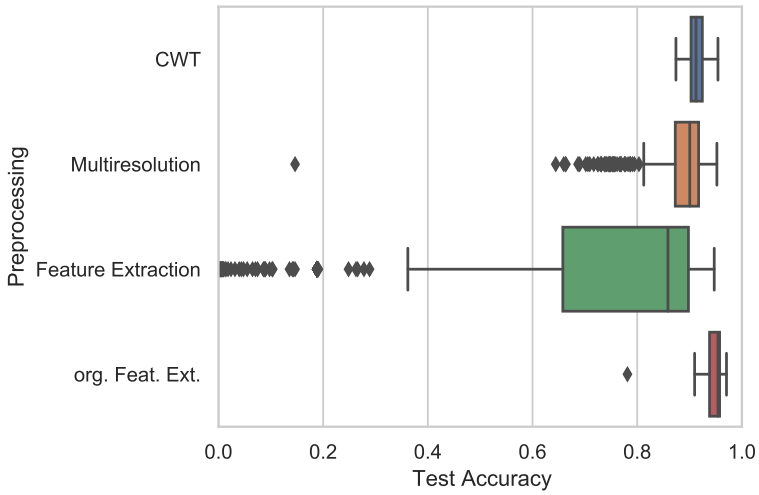

Figure 5.3: Model accuracy aggregated per major preprocessing.

closer look at which motions, e.g. different types of walking, differentiate the good and bad performing models can be found in Figures B.1 and B.2 in the appendix. A more detailed look at the results for each major preprocessing is given in the next sub-sections. The median and interquartile distance for Figure 5.1 and 5.3 is given in Tables B.1 and B.2 in the appendix respectively.

5.1.1 CWT image models

As can be observed from Figure 5.3 or the corresponding Table B.2, the convolutional neural network models using the image representation from a continuous wavelet transformation have a high median accuracy of 0.9123 with an interquartile distance of 0.0215. From Table 5.2 it can be seen that the median of the CNNs using the scaleogram

CWT wavelet	Max. CWT Scale	Median accuracy	Interquartile distance
Mexican Hat	64	0.914888	0.019817
Morlet	128	0.914634	0.021977
Gaussian-2	64	0.914126	0.021596
Gaussian-2	128	0.912348	0.011179
Morlet	64	0.909807	0.024009
Mexican Hat	128	0.907774	0.027439

Table 5.2: Aggregated CNN results per CWT wavelet and maximum scale of the CWT.

of the signals is stable with regard to the wavelet and maximum scale chosen for the

continuous wavelet transformation.

What does seem to have a higher influence is the choice of denoising method as can be seen in Figure B.3 in the appendix. The trend seen in Figure 5.2 is also present for CNN models where the median performance of the models decreases with discrete wavelet denoising compared to no denoising at all. The wavelet denoising has a high amount of variability from hyperparameters. The choice of the mother wavelet and the thresholding of the coefficients would require an extensive hyperparameter search or knowledge about the properties of the signal. The denoising without hyperparameters like median filtering could therefore be an approach that not only decreases the number of parameters that would need tuning but also decreases the variability of model performance as can be seen in Figure B.3 and Table B.4 in the appendix. From the performance of the CNN models can be followed that this approach could be an adequate replacement for the traditionally used feature based approach. Another factor is the relatively low number of hyperparameters of the CWT, wavelet family and maximum scale, compared to the feature extraction preprocessing. The results for the time-series models in the next sub-section confirm the observed low variability of the CNN models seen in this sub-section.

5.1.2 Time-series models

Another possible major preprocessing method for time-series classification is the use of coefficients from a discrete wavelet transformation. The baseline performance of the models is analysed with the time-series data and its denoising variations. This comes with distinctly less computational cost in the preprocessing compared to the scaleogram image variant. This is because of the redundancy in the continuous wavelet transformation which is used to generate the scaleogram images. For example, uncompressed arrays of the data preprocessed with CWT and a maximum scale of 128 need $\sim 6GB$ of space. The same data preprocessed with multiresolution analysis only needs $\sim 50MB$ of space. The direct use of the signals also allows the use of various other models in addition to a CNN with 1-dimensional convolutional layers which can be seen in Table 5.3. The 1-dimensional CNN has a comparable median accuracy while using data that can be stored with a fraction of space necessary for CWT.

From Table 5.3 and its corresponding boxplot in Figure B.5 in the appendix can be observed that the CNN is the model that is the least influenced by the preprocessing methods. It is also the model that has the best median accuracy followed by the LSTM-CNN mixed model from Karim et al. (2018). It can also be observed that the addition of simple bidirectionality to the RNN model improves its accuracy by ~ 0.12 and additionally

Model	Median Accuracy	Interquartile distance
CNN	0.913872	0.016514
LSTM-CNN	0.911839	0.028201
NN	0.906250	0.040777
Bi-RNN	0.890244	0.025661
RNN	0.776931	0.129954

Table 5.3: Median accuracy of the models using the raw time-series or a DWT coefficient representation. Results are aggregated over all preprocessing variations. The corresponding boxplot can be seen in Figure B.5 in the appendix.

lowers its variability due to preprocessing. The results aggregated by model and multiresolution variation can be seen in Table B.5 in the appendix. The use of raw signals requires less preprocessing than CWT transformation or the popular feature based approach seen in the next sub-section.

5.1.3 Feature models

The use of features is a preprocessing step often used in human activity recognition (see Anguita et al. (2013), Khan et al. (2010) and Yang et al. (2008)). A downside is that features beyond standard measures, like quantiles, mean and variance, often need domain knowledge, e.g. signal magnitude area for HAR which is computed as the normalized integral of an accelerometer. The use of features based on the wavelet transformation is similar to Fourier transformation as they both contain information about the frequencies present in the signal and has been widely used for time-series classification (see Farooq and Datta (2003), Xu et al. (2009) and Chen and Shen (2017)). The choice of the wavelet for the transformation is a hyperparameter that further introduces variability. The results for the models seen in Table 5.4 and are aggregated over all denoising variations and the different wavelets chosen for the wavelet based features.

From the results in Table 5.4 it can be observed that tree based ensemble machine learning models produce stabler results than the neural network model. It is also observable that the traditional statistical model, the multinomial regression, outperforms all other feature based models in the median accuracy while maintaining comparable stability as can be seen from its low interquartile distance. The model first used to analyse this dataset, the support vector classifier, in Anguita et al. (2013) is more susceptible to preprocessing than other, tree based, machine learning models or the stable implementation of the neural network. The results grouped by model and feature base are found in Table

Model	Median Accuracy	Interquartile distance
multinom. Reg.	0.904726	0.025152
GBC	0.896087	0.021087
RFC	0.893801	0.021214
NN	0.877541	0.043826
DTC	0.832063	0.028201
SVC	0.802337	0.137703
KNC	0.608232	0.137322
Sklearn NN	0.188008	0.190549

Table 5.4: Aggregated results of the models using features as input. Results are grouped by model and aggregated over denoising variations and feature bases. Feature bases contain features from the signals themselves or from the DWT coefficients.

B.6 in the appendix. The performance of the models using the original features for the dataset published with Reyes-Ortiz et al. (2016) can be seen in Table 5.5.

Model	Accuracy
multinom. Reg.	0.970573
SVC	0.960426
RFC	0.956367
NN	0.955860
Sklearn NN	0.952816
GBC	0.947742
DTC	0.909691
KNC	0.781329

Table 5.5: Classification accuracy of the models on the test set with the original features from Reyes-Ortiz et al. (2016) and Anguita et al. (2013).

Observable from Table 5.5 is that the SVC reproduces the accuracy of 0.96 or 96% from Anguita et al. (2013) and outperforms the other machine and deep learning models. Also observable is the fact that the multinomial regression again achieves the highest classification accuracy. This and the low interquartile distance used to measure susceptibility to preprocessings might make it a preferable alternative to the machine and deep learning models. Table 5.5 also shows that features build with domain knowledge outperform the other approaches.

5.2 Forecasting

The forecasting task uses the monthly data of the M4 competition dataset. The model performance is evaluated on the test set, which is 10% of the data, using the average root mean squared error(RMSE) and average symmetric mean absolute percentage error(sMAPE). The RMSE of the test set is defined as

$$RMSE(\hat{y}) = \frac{1}{N} \cdot \sum_{i=1}^{N} \sqrt{\frac{\sum_{t=1}^{T} (y_{it} - \hat{y}_{it})^2}{T}} \tag{57}$$

where $\hat{y}$ are the predictions for the test set with dimensionality $N \times T$ and $\hat{y}_{it}$ is one such prediction. The RMSE is also the metric used to monitor the deep learning models and closely related to the mean squared error which is used as loss for both machine and deep learning models. Therefore a second metric is used to measure the predictions, the sMAPE. The sMAPE is also used as evaluation metric for the M4 competition (Makridakis et al., 2020). The sMAPE is an accuracy measure based on relative errors and defined as

$$sMAPE(\hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{T} \sum_{t=1}^{T} \frac{|y_{it} - \hat{y}_{it}| \cdot 200}{|y_{it}| + |\hat{y}_{it}|} \tag{58}$$

where the 200 is used to generate percentage values and the theoretical best value is 0.

The preprocessing methods used for the forecasting include DWT denoising and multiresolution analysis based on the DWT coefficients. The wavelet based denoising reduces DWT coefficients to zero or shrinks them depending on the threshold. This alters the coefficients and therefore also the time-series if the transformation is inverted. The inverse of this alteration of the time-series cannot be applied during inference. As such it increases the forecasting bias as the model is trained to predict an altered version of the time-series. The method is based on the assumption that the time-series is made up of a "true" time-series and additional noise. The model is then able to predict this "true" time-series instead without the interference of the noise (Alrumaih and Al-Fawzan, 2002). As seen in the classification the hyperparameters are the choice of wavelet and the threshold for a soft-thresholding approach. The multiresolution analysis is a reversible method in which the DWT coefficients of the target series are directly forecast using the DWT coefficients from the input series. The inverse DWT is then applied to the coefficients which generates the prediction for the target series and allows evaluation. This method uses a single model for each of the coefficients, e.g. the coefficients cA_2, cD_2 and cD_1 for a DWT with two levels. The hyperparameters for the multiresolution analysis are the level for the DWT and the wavelet families. Both DWT denoising and multiresolution analysis use the wavelet families $\psi \in \{\text{Haar, Daubechies-4, Symlets-4, Coiflets-4}\}$ based

on Li and Tam (2017), Sugiartawan et al. (2017), Yan and Ouyang (2018) and Mishra et al. (2020).

The models are based on either the baseline deep learning models used in the M4 competition or by Mishra et al. (2020). The models include 3 machine learning models, a decision tree regression(DTR), a random forest regression(RFR) and a support vector regression(SVR). The machine learning models all use the implementation from the *scikit-learn* python library. A drawback of the implementation is that the SVR in the *scikit-learn* library is unable to predict multiple outputs. Therefore the additional *multioutputregressor* function is used to fit a SVR for each target value which are then jointly optimized. The deep learning models include a neural network(NN) with two dense layers and a convolutional neural network(CNN) with two convolutional layers followed by a single max. pooling and dense layer. A recurrent neural network(RNN) is also used which uses a single recurrent layer with LSTM units, a dropout layer with a dropout rate of 0.5 and a single dense layer. A performance baseline model is the ARIMA for which the best parameters for each time-series are chosen with an auto-arima from the python library *pmdarima*. The median results and variability over the different preprocessings can be seen in Table 5.6.

Model	Median RMSE	RMSE IQD	Median sMAPE	sMAPE IQD
RFR	1243.303247	132.461305	12.454857	2.221169
NN	1282.674080	17.549973	13.675153	0.255513
CNN	1308.555314	65.310822	13.786279	0.815363
DTR	1815.114725	366.056599	17.738360	4.563733
SVR	2382.547132	91.167948	23.192106	2.067885
RNN	5785.171263	192.790315	123.795911	21.483789

Table 5.6: Results of the forecasting models aggregated over all preprocessing variations.

From Table 5.6 it can be seen that the model with the lowest median RMSE and sMAPE is the random forest model. Although it is better performing in both metrics than all other models, it has more variability than the neural network or the convolutional neural network. The interquartile distance of the NN and CNN suggest that they are both less dependent on the preprocessing than the random forest. The tree based machine learning models again outperform the support vector model as was also the case in the classification task. The very high metric scores of the RNN in Table 5.6 are attributed to slower convergence than other deep learning models across all preprocessing

variations. A much higher number of epochs would possibly allow the RNN to improve to comparable levels to the other deep learning models.

The median performance and variability per model and major preprocessing combination can be seen in Table 5.7. It is observable from the results in the table that the random forest regression, the convolutional neural network and the neural network model all perform better with the multiresolution analysis as preprocessing. The wavelet denoising preprocessing method increases the median RMSE and sMAPE of these three models. It also increases their variability as can be seen by the increase in the interquartile distance. It is also observable from Table 5.7 that this positive influence of multiresolution analysis on the performance and variability reverses for the support vector regression and the recurrent neural network. The SVR and denoising combination also has the lowest variability of all models.

Model	Maj. Prep.	median RMSE	RMSE IQD	median sMAPE	sMAPE IQD
RFR	Multiresolution	1232.994632	5.680089	12.275056	0.196409
CNN	Multiresolution	1264.508698	51.112332	13.277548	0.775055
NN	Multiresolution	1277.785375	7.858523	13.457280	0.095140
	Denoising	1290.280206	28.727466	13.680545	0.209592
CNN	Denoising	1309.762281	62.754894	14.108509	1.374528
RFR	Denoising	1335.539991	160.672735	13.985533	2.608262
DTR	Multiresolution	1793.179541	56.986714	17.403203	0.366574
	Denoising	2150.641037	585.026244	21.322216	5.515836
SVR	Denoising	2377.796901	4.151135	23.114970	0.088712
	Multiresolution	2536.244707	129.432358	26.415080	2.487325
RNN	Denoising	5780.957460	101.213565	123.368875	9.688638
	Multiresolution	6008.797750	205.553907	150.318774	28.505949

Table 5.7: Median RMSE and sMAPE and their respective IQD over model and preprocessing combination. Results are sorted by ascending median RMSE.

Table 5.8 contains the baseline results for each model and the additional baseline for the ARIMA model. The baseline results were calculated using the raw data as input. ARIMA results are averaged over the test set and one model is fit for each of the time series. From Table 5.8 it can be seen that the ARIMA model outperforms the machine and deep learning models in the absolute error measure while having a much higher relative

Model	RMSE	sMAPE
ARIMA	738.736224	77.078693
RFR	1243.303247	12.143546
NN	1276.111456	13.675154
CNN	1278.732336	13.506073
DTR	1806.965950	16.816696
SVR	2377.796901	23.091703
RNN	5683.342951	114.045341

Table 5.8: Results of the various forecasting models using the raw data as input. The ARIMA baseline performs better for the absolute error measure and worse for the relative error measure.

error. This indicates that the ARIMA model fails to predict low values in the time-series for which the relative error is more sensitive. It is also observable that the performance trend visible in Table 5.7 is repeated for forecasts using the raw data as input.

6 Conclusion

In this thesis, the variability of different deep and machine learning models due to preprocessing is investigated on two different time-series tasks. The classification data is preprocessed with denoising and either a continuous wavelet transformation, a multiresolution analysis or feature extraction. These preprocessings each come with variability and influence models in different ways. The results show that convolutional neural networks are the model with the least variability due to preprocessing and therefore the most stable of the analysed models. The classification results show that tree based ensemble machine learning models, like the random forest or gradient boosting model, are less influenced by preprocessing variations than the simple decision tree, a k-nearest neighbour classifier or a support vector classifier. Also observable from the results is that the addition of simple bidirectionality to the recurrent neural network lowered the variability due to preprocessing while also increasing the median classification accuracy. The results based on the raw signals showed that a simple transformation in the frequency domain with a discrete wavelet transformation can improve the classification accuracy while lowering the variability. The results for the feature based approach show that features based on the discrete wavelet transformation coefficients are preferable to features based on the signals themselves if those features were constructed without domain knowledge. The wavelet based features are also more robust with regards to any prior denoising of the data. The results of the classification preprocessing variations also show that the best performance is achieved with carefully chosen domain based features. Future work would further investigate the multiresolution approach as it comes with lower computational cost and is applicable to a wide range of time-series without prior domain knowledge.

The forecasting results show that while the random forest model achieves better median results, it is also more susceptible to preprocessing variations than a convolutional neural network or neural network.

The results in this thesis indicate that no denoising is better for human activity recognition tasks than too strong denoising. The results indicate that the parameterless approach of median denoising is a good alternative. It is also less dependent on the data as the wavelet denoising. The wavelet denoising also introduces multiple new hyperparameters which would need tuning for a new dataset and increase the general variability of the method. The further direction of this work would either be the study of the preprocessing methods on other datasets or further variations in preprocessing that were not explored in this thesis, e.g. restriction of the input signals to either acceleration or angular velocity signals. Future work would also apply the preprocessing methods to other datasets as

the denoising variability might be caused by the nature of the human activity recognition data. The results can vary for the same model and preprocessing due to variability during model training. Further work would also minimize this variability with cross-validation.

Bibliography

References

Aggarwal, C. C. and Yu, P. S. (2001). Outlier detection for high dimensional data. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 37–46.

Ahmed, N. K., Atiya, A. F., Gayar, N. E., and El-Shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621.

Alrumaih, R. M. and Al-Fawzan, M. A. (2002). Time series forecasting using wavelet denoising an application to saudi stock index. *Journal of King Saud University - Engineering Sciences*, 14(2):221–233.

AlSharabi, K., Ibrahim, S., Djemal, R., and Alsuwailem, A. (2016). A dwt-entropy-ann based architecture for epilepsy diagnosis using eeg signals. In *2016 2nd international conference on advanced technologies for signal and image processing (ATSIP)*, pages 288–291. IEEE.

Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. (2013). A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, page 3.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, NY. Softcover published in 2016.

Blu, T. (1993). Iterated filter banks with rational rate changes connection with discrete wavelet transforms. *IEEE Transactions on Signal Processing*, 41(12):3232–3244.

Böhning, D. (1992). Multinomial logistic regression algorithm. *Annals of the institute of Statistical Mathematics*, 44(1):197–200.

Boshnakov, G. N. (2011). On first and second order stationarity of random coefficient models. *Linear Algebra and its Applications*, 434(2):415–423.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Camacho-Collados, J. and Pilehvar, M. T. (2017). On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. *arXiv preprint arXiv:1707.01780*.

Chandrashekar, G. and Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28.

Chen, Y. and Shen, C. (2017). Performance analysis of smartphone-sensor behavior for human activity recognition. *IEEE Access*, 5:3095–3110.

Chollet, F. et al. (2015). Keras. https://keras.io.

Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling.

Conejo, A. J., Plazas, M. A., Espinola, R., and Molina, A. B. (2005). Day-ahead electricity price forecasting using the wavelet transform and arima models. *IEEE transactions on power systems*, 20(2):1035–1042.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301.

Coppersmith, D., Hong, S. J., and Hosking, J. R. (1999). Partitioning nominal attributes in decision trees. *Data Mining and Knowledge Discovery*, 3(2):197–217.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Dagum, E. B. (1980). *The X-II-ARIMA seasonal adjustment method*. Statistics Canada, Seasonal Adjustment and Time Series Staff.

Daubechies, I. (1992). *Ten lectures on wavelets*. SIAM.

Donoho, D. L. and Johnstone, I. M. (1994). Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Evrendilek, F. (2014). Modeling net ecosystem carbon dioxide exchange using temporal neural networks after wavelet denoising. *Geographical Analysis*, 46(1):37–52.

Eyben, F., Wöllmer, M., and Schuller, B. (2010). Opensmile: the munich versatile and fast open-source audio feature extractor. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1459–1462.

Farooq, O. and Datta, S. (2003). Phoneme recognition using wavelet based features. *Information Sciences*, 150(1-2):5–15.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

García, S., Luengo, J., and Herrera, F. (2015). *Data preprocessing in data mining*. Springer.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385. Springer Science & Business Media.

Grochowski, M. and Jankowski, N. (2004). Comparison of instance selection algorithms ii. results and comments. In *International Conference on Artificial Intelligence and Soft Computing*, pages 580–585. Springer.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Higham, C. F. and Higham, D. J. (2019). Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

Hodge, V. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Jankowski, N. and Grochowski, M. (2004). Comparison of instances seletion algorithms i. algorithms survey. In *International conference on artificial intelligence and soft computing*, pages 598–603. Springer.

Karim, F., Majumdar, S., Darabi, H., and Chen, S. (2018). Lstm fully convolutional networks for time series classification. *IEEE Access*, 6:1662–1669.

Keller, J. M., Gray, M. R., and Givens, J. A. (1985). A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(4):580–585.

Khan, A. M., Lee, Y.-K., Lee, S.-Y., and Kim, T.-S. (2010). Human activity recognition via an accelerometer-enabled-smartphone using kernel discriminant analysis. In *2010 5th international conference on future information technology*, pages 1–6. IEEE.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. (2006). Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117.

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Handwritten digit recognition with a back-propagation network. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, pages 396–404.

Lee, G. R., Gommers, R., Waselewski, F., Wohlfahrt, K., and O'Leary, A. (2019). Pywavelets: A python package for wavelet analysis. *Journal of Open Source Software*, 4(36):1237.

Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Li, Z. and Tam, V. (2017). Combining the real-time wavelet denoising and long-short-term-memory neural network for predicting stock indexes. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE.

Liu, H., Tian, H.-q., Pan, D.-f., and Li, Y.-f. (2013). Forecasting models for wind speed using wavelet, wavelet packet, time series and artificial neural networks. *Applied Energy*, 107:191–208.

Madine, M., Battah, A., Khan, A., and Werghi, N. (2019). Detecting drivers smartphone: A learned features approach using aggregated scalogram images. In *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–5. IEEE.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2020). The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74. M4 Competition.

Mallat, S. G. (1989). A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11(7):674–693.

Mishra, S., Bordin, C., Taharaguchi, K., and Palu, I. (2020). Comparison of deep learning models for multivariate prediction of time series wind power generation and temperature. *Energy Reports*, 6:273–286. Technologies and Materials for Renewable Energy, Environment and Sustainability.

Mörchen, F. (2003). Time series feature extraction for data mining using dwt and dft.

Naduvil-Vadukootu, S., Angryk, R. A., and Riley, P. (2017). Evaluating preprocessing strategies for time series prediction using deep learning architectures. In *The Thirtieth International Flairs Conference*.

Nason, G. P., Von Sachs, R., and Kroisandt, G. (2000). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(2):271–292.

Ng, A. Y. (2004). Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Percival, D. B. and Walden, A. T. (2000). *Wavelet methods for time series analysis*, volume 4. Cambridge university press.

Renaud, O., Starck, J.-L., and Murtagh, F. (2005). Wavelet-based combined signal filtering and prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(6):1241–1251.

Reyes-Ortiz, J.-L., Oneto, L., Samà, A., Parra, X., and Anguita, D. (2016). Transition-aware human activity recognition using smartphones. *Neurocomputing*, 171:754–767.

Richman, J. S. and Moorman, J. R. (2000). Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology-Heart and Circulatory Physiology*.

Saeys, Y., Inza, I., and Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517.

Schlüter, S. and Deuschle, C. (2010). Using wavelets for time series forecasting: Does it pay off? Technical report, IWQW Discussion Papers.

Sidney Burrus, C., Gopinath, R. A., and Guo, H. (1998). Introduction to wavelets and wavelet transforms. *A Primer; Prentice Hall: Upper Saddle River, NJ, USA*.

Smith, T. G. et al. (2017–). pmdarima: Arima estimators for Python. [Online; accessed 20.02.2021].

Spiliotis, E., Kouloumos, A., Assimakopoulos, V., and Makridakis, S. (2020). Are forecasting competitions data representative of the reality? *International Journal of Forecasting*, 36(1):37–53. M4 Competition.

Sugiartawan, P., Pulungan, R., and Sari, A. K. (2017). Prediction by a hybrid of wavelet transform and long-short-term-memory neural network. *International Journal of Advanced Computer Science and Applications*, 8(2):326–332.

Tabik, S., Peralta, D., Herrera-Poyatos, A., and Herrera, F. (2017). A snapshot of image pre-processing for convolutional neural networks: case study of mnist. *International Journal of Computational Intelligence Systems*, 10(1):555–568.

Valens, C. (1999). A really friendly guide to wavelets. [Online; accessed 20.02.2021].

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Xu, Q., Zhou, H., Wang, Y., and Huang, J. (2009). Fuzzy support vector machine for classification of eeg signals using wavelet-based features. *Medical engineering & physics*, 31(7):858–865.

Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629.

Yan, H. and Ouyang, H. (2018). Financial time series prediction based on deep learning. *Wireless Personal Communications*, 102(2):683–700.

Yang, J.-Y., Wang, J.-S., and Chen, Y.-P. (2008). Using acceleration measurements for activity recognition: An effective learning algorithm for constructing neural classifiers. *Pattern recognition letters*, 29(16):2213–2220.

Yim, J. and Sohn, K.-A. (2017). Enhancing the performance of convolutional neural networks on quality degraded datasets. In *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8. IEEE.

Zheng, X., Wang, M., and Ordieres-Meré, J. (2018). Comparison of data preprocessing approaches for applying deep learning to human activity recognition in the context of industry 4.0. *Sensors*, 18(7):2146.

Zhou, Y.-T. and Chellappa, R. (1988). Computation of optical flow using a neural network. In *ICNN*, pages 71–78.

A Appendix Methodology

A.1 LSTM-CNN architecture

The model introduced in Karim et al. (2018) contains as mentioned in Sub-Section 3.1.5 the additional batch norm and dropout layers. The full model architecture can be seen in Figure A.1.

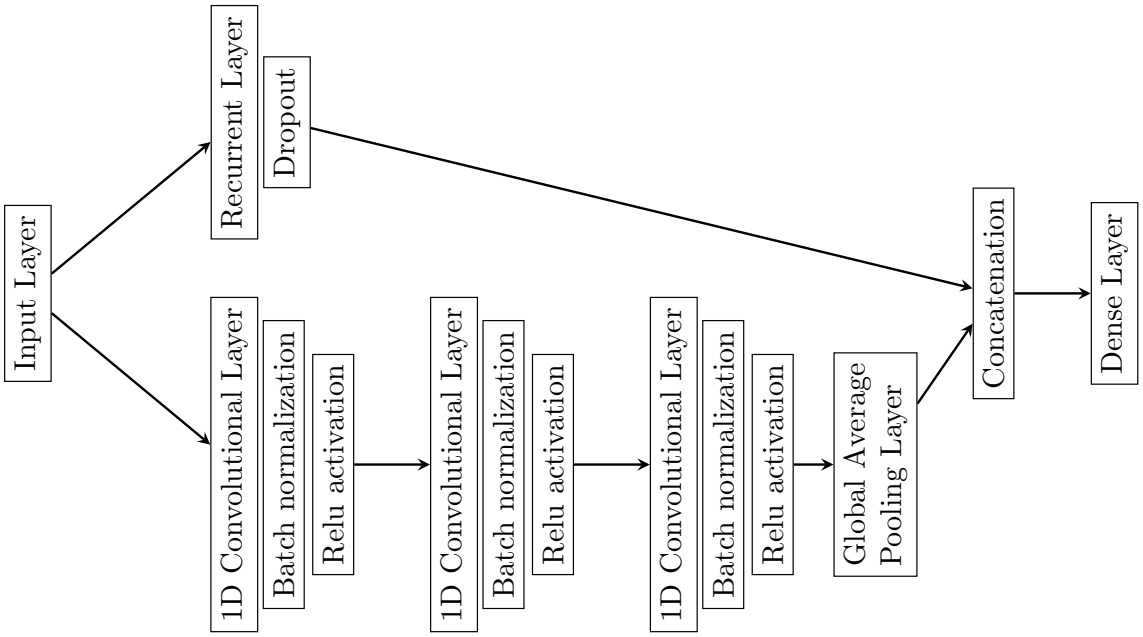

Figure A.1: Architecture of the LSTM-CNN model from Karim et al. (2018) used for the multiresolution classification. Dropout is quite aggressive with 0.8 dropout rate.

A.2 Continuous Wavelets

Continuous wavelet	Notation
Mexican Hat Wavelet	$\psi(t) = \frac{2}{\sqrt{3}\sqrt[4]{\pi}} \exp^{-\frac{t^2}{2}} \left(1-t^2\right)$
Morlet Wavelet	$\psi(t) = \exp^{-\frac{t^2}{2}} \cos(5t)$
Gaussian Derivative Wavelets	$\psi(t) = C_p \exp^{-t^2}$

Table A.1: Continuous wavelets used in this thesis. C_p in the Gaussian Derivative Wavelets is the order dependent normalization constant and is given as $\|f^{(p)}\|^2 = 1$ where $f^{(p)}$ is the p^{th} order derivative of f, here $p = 2$.

A.3 Discrete Wavelets

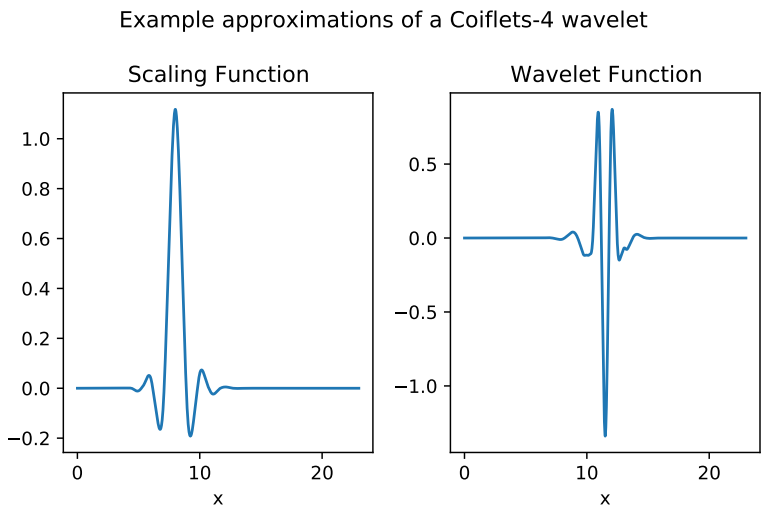

Figure A.2: Example approximations of the scaling and wavelet functions of a Coiflet-4 wavelet used in this thesis.

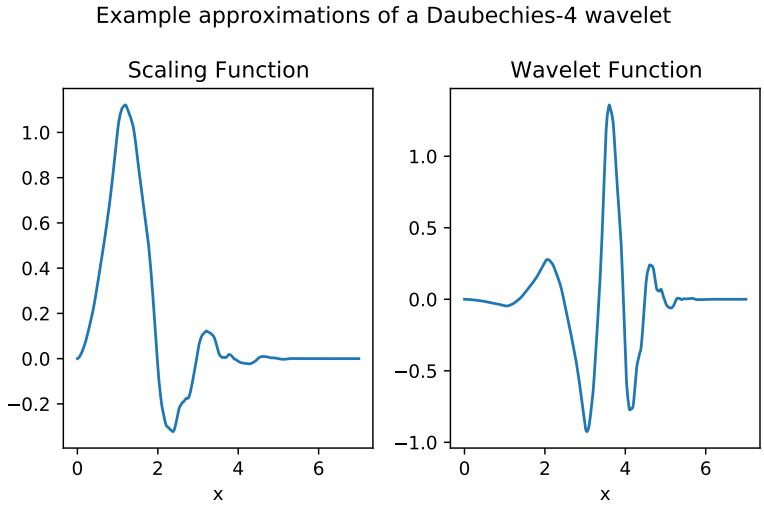

Figure A.3: Example approximations of the scaling and wavelet functions of a Daubechies-4 wavelet used in this thesis.

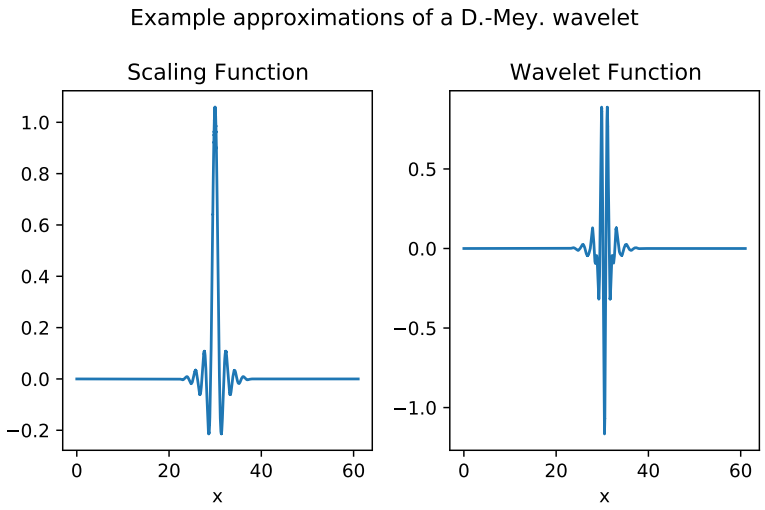

Figure A.4: Example approximations of the scaling and wavelet functions of a discrete approximation of a Meyer wavelet used in this thesis.

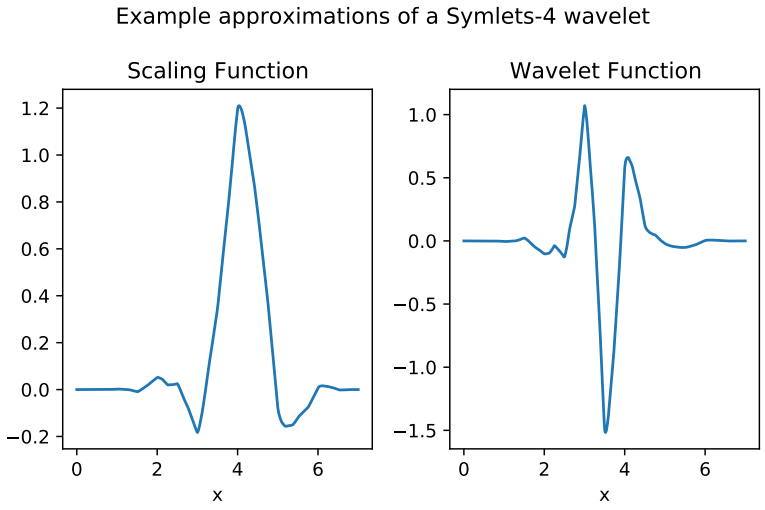

Figure A.5: Example approximations of the scaling and wavelet functions of a Symlets-4 wavelet used in this thesis.

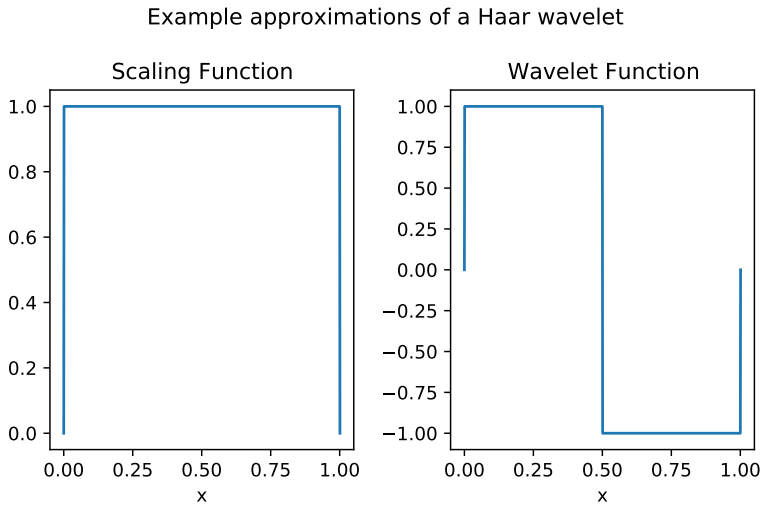

Figure A.6: Example approximations of the scaling and wavelet functions of a Haar wavelet used in this thesis.

B Appendix Experiments

Model	Median accuracy	Interquartile distance
CNN	0.913364	0.018801
LSTM-CNN	0.911839	0.028201
multinom. Reg.	0.905996	0.025915
GBC	0.896341	0.021341
RFC	0.893801	0.023882
Bi-RNN	0.890244	0.025661
NN	0.889736	0.042175
DTC	0.832317	0.028963
SVC	0.803862	0.138211
RNN	0.776931	0.129954
KNC	0.611280	0.145833
Sklearn NN	0.188008	0.193598

Table B.1: Median accuracy and interquartile distance of the models used for classification aggregated by model.
Results are aggregated over all preprocessing variations that the model was used for.

Major Preprocessing	Median accuracy	Interquartile distance
original Features	0.954338	0.019153
CWT	0.912348	0.021596
Multiresolution	0.900406	0.044588
Feature Extraction	0.858740	0.239964

Table B.2: Median accuracy and interquartile distance of the models used for classification aggregated by major preprocessing.
Results are aggregated over all models used for a major preprocessing.

Observable from Figure B.1 is that the main positions are accurately predicted, especially the distinction between walking. The main hurdle for further classification improvements is the distinction between sitting and standing. Worse models struggle with the distinction between the different types of walking as can be seen in Figure B.2.

DWT denoising wavelet	Median accuracy	Interquartile distance
Median	0.920732	0.071138
none	0.917429	0.059832
Coiflets-4	0.891514	0.088161
Daubechies-4	0.886179	0.110010
Symlets-4	0.885671	0.105564
D.-Mey.	0.883892	0.120681
Haar	0.874238	0.128176

Table B.3: Aggregated accuracy over all models per denoising wavelet for the classification task.

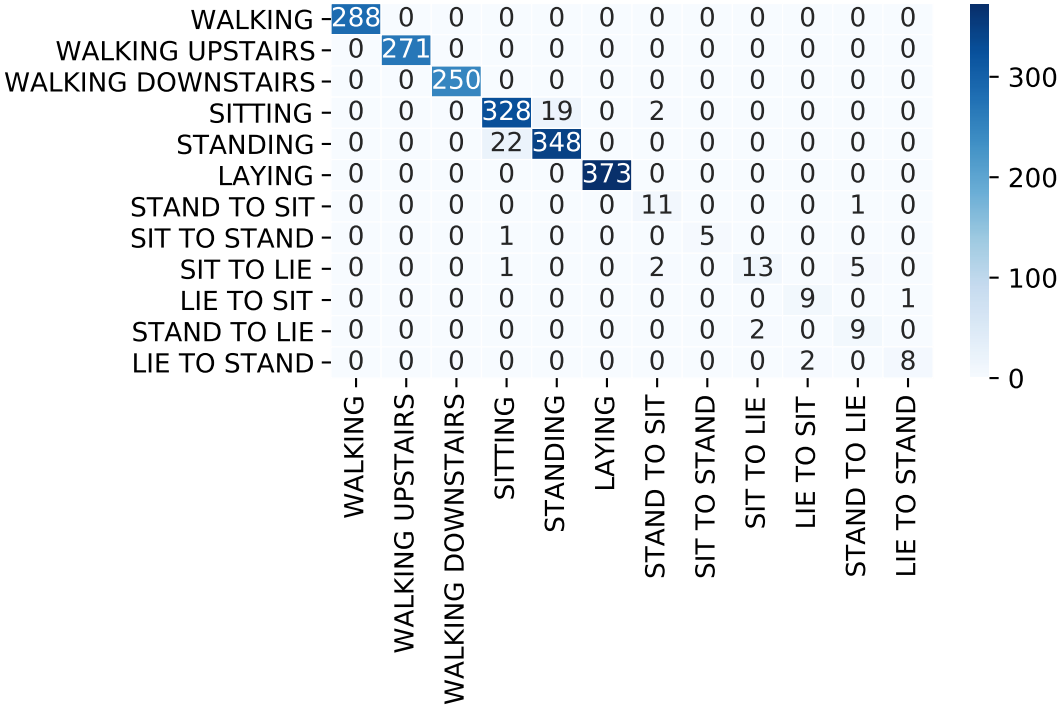

Figure B.1: Confusion matrix for the multinomial regression with the original features. The best performing model which can accurately differentiate the types of walking. Main improvement is the distinction between sitting and standing. The influence of accurately classified transitions is minimal due to the relatively short time spans in which they are performed. Accuracy = 0.970573.

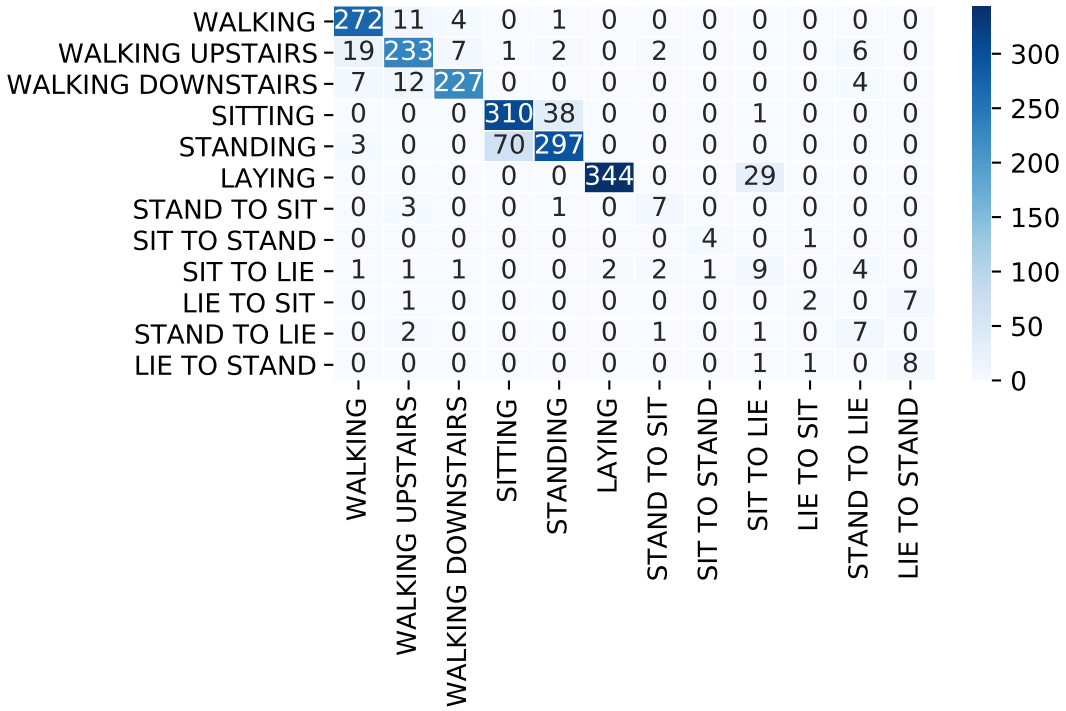

Figure B.2: Confusion matrix for the worst CNN model with CWT as major preprocessing. Preprocessing parameters are: Denoising wavelet = Haar, Threshold = 0.3, Continuous wavelet = Mexican Hat, Max. Scale for the CWT = 128. Observable is the miss-classification of different walking distinctions. Accuracy = 0.873984.

B.1 CWT results

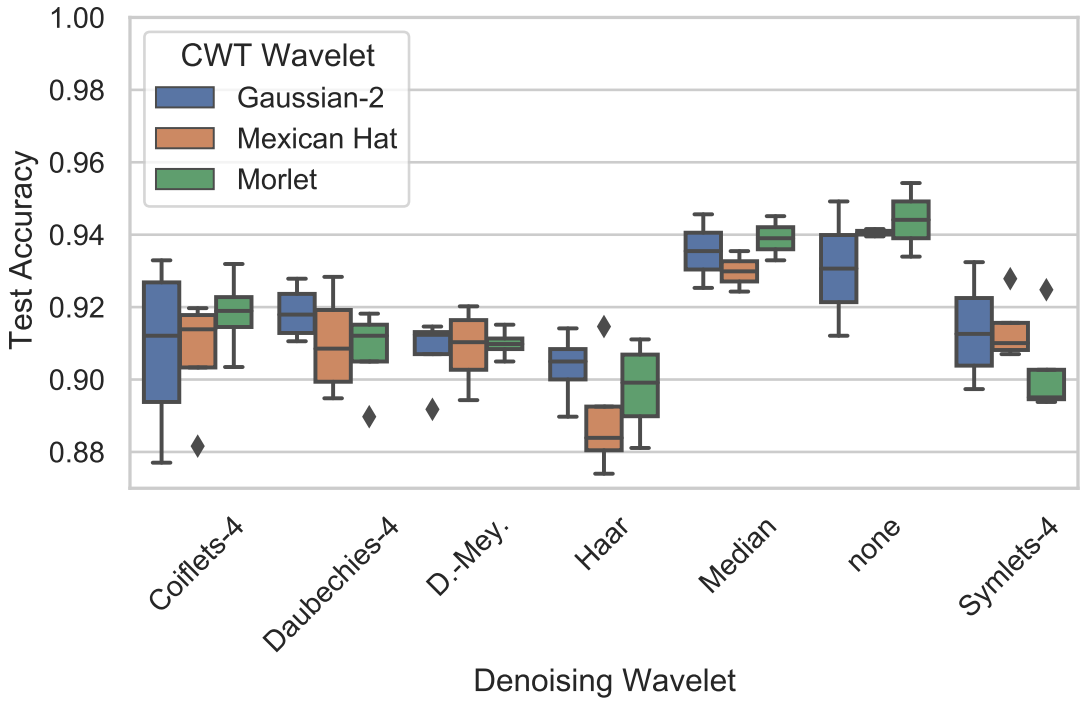

Figure B.3: Aggregated results for CWT scaleogram as major preprocessing split by denoising method and CWT wavelet. A notable observation is the performance of the CNN without denoising or median denoising.

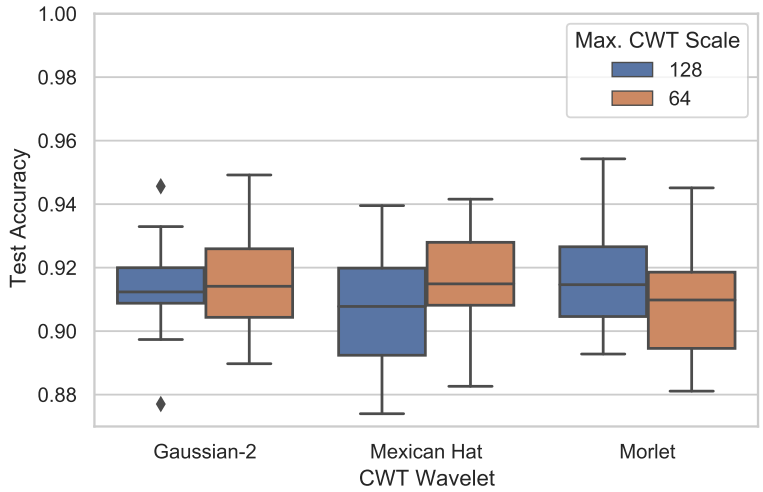

Figure B.4: Aggregated results for CWT scaleogram as major preprocessing split by chosen CWT wavelet and maximal scale for the CWT.

CWT wavelet	DWT denoising wavelet	Median accuracy	Interquartile distance
Morlet	none	0.944106	0.010163
Mexican Hat	none	0.940549	0.001016
Morlet	Median	0.939024	0.006098
Gaussian-2	Median	0.935467	0.010163
	none	0.930640	0.018547
Mexican Hat	Median	0.929878	0.005589
Morlet	Coiflets-4	0.918953	0.008257
Gaussian-2	Daubechies-4	0.917937	0.010798
Mexican Hat	Coiflets-4	0.913872	0.014482
Gaussian-2	Symlets-4	0.912602	0.018674
	D.-Mey.	0.912348	0.006098
Morlet	Daubechies-4	0.912094	0.010163
Gaussian-2	Coiflets-4	0.912093	0.033028
Mexican Hat	D.-Mey.	0.910315	0.013720
	Symlets-4	0.910061	0.007495
Morlet	D.-Mey.	0.909807	0.002922
Mexican Hat	Daubechies-4	0.908537	0.019817
Gaussian-2	Haar	0.904980	0.008384
Morlet	Haar	0.899136	0.017022
	Symlets-4	0.895071	0.008130
Mexican Hat	Haar	0.883892	0.012068

Table B.4: Median accuracy of the CNN models using the scaleogram images from CWT as input.

B.2 Multiresolution TS

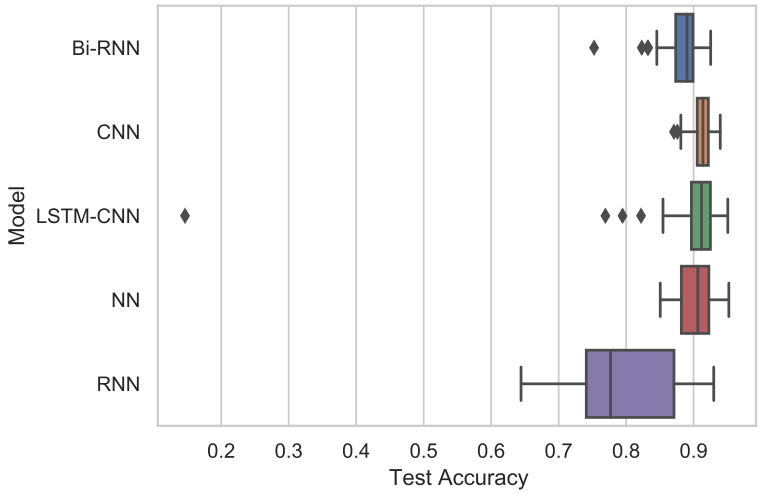

Figure B.5: Aggregated results for the multiresolution classification models grouped by model and aggregated over all preprocessing steps.

Model	Multires. wavelet	Median accuracy	Interquartile distance
CNN	Haar	0.923781	0.007495
NN	Haar	0.923018	0.011179
LSTM-CNN	none	0.922764	0.018674
NN	Daubechies-4	0.917429	0.015117
CNN	Daubechies-4	0.916667	0.011433
LSTM-CNN	D.-Mey.	0.916413	0.022485
CNN	Symlets-4	0.915396	0.009909
NN	Symlets-4	0.914888	0.026042
LSTM-CNN	Haar	0.914380	0.035061
CNN	D.-Mey.	0.914126	0.021850
LSTM-CNN	Coiflets-4	0.910569	0.027185
CNN	Coiflets-4	0.908283	0.014863
LSTM-CNN	Daubechies-4	0.905742	0.036077
CNN	none	0.904472	0.015879
Bi-RNN	D.-Mey.	0.897612	0.016641
	Haar	0.895325	0.015752
LSTM-CNN	Symlets-4	0.892530	0.035315
Bi-RNN	Symlets-4	0.890752	0.017149
	Coiflets-4	0.890244	0.014101
NN	Coiflets-4	0.888719	0.026169
	D.-Mey.	0.887957	0.021087
RNN	none	0.886687	0.033028
Bi-RNN	Daubechies-4	0.880843	0.021850
NN	none	0.878557	0.033791
Bi-RNN	none	0.871697	0.016768
RNN	Haar	0.833841	0.035442
	Daubechies-4	0.767530	0.032012
	D.-Mey.	0.765244	0.034553
	Coiflets-4	0.739583	0.022358
	Symlets-4	0.697917	0.074441

Table B.5: Median accuracy of Multiresolution results grouped by model and wavelet used in the multiresolution splitting. Aggregated over all denoising variations.

B.3 Feature models

Model	DWT wavelet	Median Accuracy	Interquartile distance
multinom. Reg.	Haar	0.914888	0.011941
	Daubechies-4	0.910823	0.014863
	Symlets-4	0.908791	0.012322
RFC	Haar	0.908537	0.012322
GBC	Haar	0.907520	0.012576
SVC	Haar	0.903709	0.019944
NN	Daubechies-4	0.902693	0.020325
Sklearn NN	Haar	0.900915	0.007749
GBC	D.-Mey.	0.900407	0.014990
multinom. Reg.	D.-Mey.	0.899644	0.005462
NN	Symlets-4	0.898882	0.017403
RFC	Symlets-4	0.898628	0.013211
GBC	Symlets-4	0.896596	0.017530
RFC	none	0.895579	0.618267
	D.-Mey.	0.894817	0.022739
GBC	Daubechies-4	0.893547	0.013084
NN	Haar	0.893293	0.017912
RFC	Daubechies-4	0.889228	0.008257
	Coiflets-4	0.888720	0.009400
GBC	Coiflets-4	0.888211	0.009273
multinom. Reg.	Coiflets-4	0.886433	0.012449
GBC	none	0.882622	0.896850
NN	Coiflets-4	0.863059	0.017530
	D.-Mey.	0.858740	0.016895
DTC	Haar	0.842226	0.022104
SVC	Daubechies-4	0.839177	0.049670
DTC	D.-Mey.	0.837398	0.022993
	Symlets-4	0.832317	0.012322
	Coiflets-4	0.831809	0.021341
	Daubechies-4	0.826982	0.021341
SVC	Symlets-4	0.821138	0.042048
DTC	none	0.782520	0.698298
KNC	Haar	0.749492	0.028074
SVC	D.-Mey.	0.733232	0.015244

Model	DWT wavelet	Median Accuracy	Interquartile distance
	Coiflets-4	0.688770	0.062246
KNC	Daubechies-4	0.626016	0.021087
	Symlets-4	0.590447	0.077617
	D.-Mey.	0.569614	0.030488
	none	0.535061	0.624238
	Coiflets-4	0.532520	0.037983
SVC	none	0.436484	0.740854
Sklearn NN	none	0.225356	0.212779
multinom. Reg.	none	0.206047	0.886306
Sklearn NN	Daubechies-4	0.188008	0.000000
	Symlets-4	0.188008	0.000000
	D.-Mey.	0.188008	0.000000
	Coiflets-4	0.188008	0.000000
NN	none	0.139482	0.714431

Table B.6: Aggregated results of the models using feature inputs. Results are grouped by model and major preprocessing variation. DWT wavelet denotes here if the features are extracted from the DWT coefficients with the corresponding wavelet or based on the signal itself if the wavelet is "none". Aggregation is over all denoising variations for the combination.

C Electronic Annex

The electronic delivery of this thesis includes:

- This thesis as PDF file in its current form,
- the source code written in Python, version 3.7.9, 64-bit (Van Rossum and Drake, 2009),
- the original datasets as: ZIP-files, unpacked as CSV and TXT files,
- the results as CSV and tex files,
- the models in either the JOBLIB or H5 format
- most of the preprocessed data in the NPZ format.

The electronic delivery does not include the data preprocessed as images through CWT as these datasets are ∼1GB and ∼6GB in size for max. scales of 64 and 128 respectively.
The code, original datasets in ZIP format and the results are also available on Github: https://github.com/joshuawagner93/master_thesis_wagner_joshua. Included is a Readme.md file which explains the steps to run the code and how to initialize a new virtual environment with the right Python module versions.

Statement of authorship

I hereby declare that I have written the master's thesis on my own without the use of any sources other than those cited in the text, graphics, tables and formulas. Neither this nor a similar work has been presented to an examination committee.

Munich, March 22, 2021 .....