Department of Physics and Astronomy

University of Heidelberg

Master's Thesis in Physics

submitted by

Arne Emmel

November 2020

Inference with Convolutional Neural Networks on Analog Neuromorphic Hardware

This thesis has been carried out by

Arne Emmel

at the

Kirchhoff-Institute for Physics

Electronic Vision(s) Group

under the supervision of

Dr. Johannes Schemmel

Abstract

In the context of a competition to advance energy efficient hardware solutions for artificial intelligence, the analog BrainScaleS-2 system is used to accelerate artificial neural networks. For this purpose, electrocardiograms are analyzed and possible classification methods are presented, including methods from chaos theory as well as convolutional neural networks. The latter are universally applicable, which is additionally shown by an application to sensor data for activity recognition. In order to realize these artificial neural networks under transparent integration of the BrainScaleS-2 system, the PyTorch extension `hxtorch` is supplemented by further functionality. In addition, a simulation mode is implemented, which uses quantization and statistical noise to represent some properties of analog inference for training hardware-compatible models. The presented methods allow detection of atrial fibrillation in electrocardiogram recordings with average runtimes down to 10 µs. Depending on the optimization target, accuracies between 92.2 % and 95.4 % are achieved on the provided dataset with false positive rates from 17.1 % to 8.7 %.

Zusammenfassung

Im Rahmen eines Wettbewerbs zur Förderung energiesparender Hardwarelösungen für künstliche Intelligenz wird das analoge neuromorphe BrainScaleS-2 System für die Beschleunigung künstlicher neuronaler Netzwerke verwendet. Dazu werden Elektrokardiogramme analysiert und mögliche Klassifizierungsmethoden aufgezeigt, dies beinhaltet sowohl Methoden aus der Chaostheorie als auch faltende neuronale Netzwerke. Letztere sind sehr universell einsetzbar, was zusätzlich durch eine Anwendung auf Sensordaten zur Aktivitätserkennung gezeigt wird. Um diese künstlichen neuronalen Netzwerke unter transparenter Einbindung des BrainScaleS-2 Systems zu ermöglichen, wird die PyTorch-Erweiterung `hxtorch` um weitere Funktionalität ergänzt. Zusätzlich wird ein Simulationsmodus implementiert, welcher mit Quantisierung und statistischem Rauschen einige Eigenschaften der analogen Inferenz abbildet, um mit der Hardware kompatible Modelle zu trainieren. Die vorgestellten Methoden ermöglichen den Nachweis von Vorhofflimmern in Elektrokardiogrammaufnahmen bei durchschnittlichen Laufzeiten bis hinunter zu 10 µs. Abhängig vom Optimierungsziel werden auf dem zur Verfügung gestellten Datensatz Genauigkeiten zwischen 92,2 % und 95,4 % bei Falsch-Positiv-Raten von 17,1 % bis 8,7 % erreicht.

Contents

1 Introduction **2**

2 Fundamentals **4**
2.1 Basics and Terminology of Machine Learning 4
2.1.1 Artificial Neural Networks 4
2.1.2 Supervised Learning 6
2.1.3 Layer Architectures 8
2.1.4 Initialization 10
2.2 The BrainScaleS-2 Neuromorphic Hardware 12
2.2.1 Matrix Multiplication on BrainScaleS-2 13
2.2.2 PyTorch Integration 15
2.3 Datasets 15
2.3.1 Electrocardiography 16
2.3.2 Human Activity Recognition 23

3 Methods and Tools **25**
3.1 ECG Preprocessing 25
3.1.1 Baseline Correction 26
3.1.2 The Difference Method 27
3.1.3 Sample Rate Conversion 27
3.1.4 Beat Detection 30
3.2 Classification Methods 32
3.2.1 Plot of Successive RR Intervals 32
3.2.2 CNN on Sparse Beat Positions 33
3.2.3 CNN on Less Preprocessed ECG Recordings 34
3.2.4 Human Activity Recognition 36

4 Software Implementation **37**
4.1 The Processing Framework 37
4.1.1 Design Principles 37
4.1.2 Components 39
4.1.3 Caching 41
4.2 `hxtorch` – PyTorch on BrainScaleS-2 43
4.2.1 Backward Pass for Training with Hardware in the Loop 43
4.2.2 The Convolution Operations 44
4.2.3 The Mock-Mode – Simulating the Hardware 46
4.2.4 Replacements for the PyTorch Layers 49

4.2.5 Initialization 52
4.2.6 Move Pretrained Models to `hxtorch` 52

5 Results **53**
5.1 ECG Classification 53
5.1.1 Beat Position Based Models 53
5.1.2 Less Preprocessing: Convolutional Neural Network Models 57
5.2 Human Activity Recognition 65

6 Discussion and Outlook **67**

Bibliography **70**

1 Introduction

With the increasing availability of small low power microprocessors, embedded inference and machine learning is becoming more and more convenient. Data can be processed closer to the sensors, saving unnecessary detours to host computers or even to the cloud, thus reducing latency, required network bandwidth, and energy consumption (Merenda, Porcaro, and Iero, 2020).

Low latency is essential for autonomous driving (Lee, Tsung, and Wu, 2018). Real-time video analysis requires high data rates and therefore benefits considerably from edge computing (Ananthanarayanan et al., 2017). Drones and other robotic applications additionally gain from low energy consumption and small dimensions. Another rapidly growing application area for embedded machine learning solutions is the processing of particularly sensitive personal data (Zhao et al., 2018). Those make it possible to process data directly on users' devices and thus protect sensitive information from access by third parties. This privacy aspect also includes two typical areas of application that are addressed in this thesis: medical applications and the monitoring of daily activities that might be useful for digital assistance and in elderly care.

Several processors specialized for embedded inference have been released recently. A prominent example is Google's edgeTPU, a low power tensor processing unit. With a performance of 4 TOP/s, i. e., four trillion fixed-point operations per second, at a power consumption of 2 W (Coral, 2020), it offers an outstanding energy efficiency. The competitor ARM, known for energy-efficient microprocessor designs, released their first microNPU (micro neural processing unit) in spring 2020 (ARM, 2020a), promising a low-cost energy efficient machine learning solution for embedded devices. The follow-up version was presented in October 2020 (ARM, 2020b) and is capable of up to 1 TOP/s.

Like the majority of solutions, these examples are based on digital circuits. Mixed-signal solutions that could achieve even higher energy efficiency have been proposed as well, e. g., by Schemmel, Hohmann, et al. (2004), Verleysen et al. (1994), and Yamaguchi et al. (2019). The data handling is still digital, but the multiply accumulate operation is realized in an analog core which accumulates the values as electric charge on a capacitor. This is much more energy efficient than the equivalent digital operation. However, corrections must be applied in this case in order to compensate for tolerances in production and the associated deviations in analog properties.

With the neuromorphic BrainScaleS-2 system such a mixed-signal solution is used in this thesis. Originally designed for the physical emulation of spiking neural networks, its versatility also makes it suitable for analog multiply accumulate operations. As in other analog systems, there are some particularities to consider, like analog noise, fixed pattern deviations and a limited resolution. However, if its accuracy is sufficient, analog computing may be a promising approach to build energy-efficient and affordable hardware

in the future, which could become particularly useful in embedded applications.

An important medical application of these systems is the detection of cardiac arrhythmias. Arrhythmias are among the most prominent complications encountered by adults with congenital heart defects and affect about 9 in 1000 live births worldwide (Linde et al., 2011). They can cause further complications like stroke, heart attack and may even lead to sudden cardiac death. Atrial fibrillation, the most common arrhythmia, can be treated effectively, e. g., by eliminating underlying causes or by providing a regular rhythm by means of drugs or a pacemaker (Wyse et al., 2002). Nevertheless, the diagnosis is not trivial as most of the arrhythmias may occur in episodes only and are not permanent. For this purpose small portable devices are used, which monitor the heart rhythm and can be worn on the body for a longer time. This data could be examined directly on the device by means of artificial intelligence.

The main part of this master thesis, the design of a model for the classification of ECG recordings, is part of the participation of the Electronic Vision(s) research group in the competition "Energieeffizientes KI-System" (German; Energy-efficient AI system), initiated by the Federal Ministry of Education and Research in Germany (BMBF, 2019). The aim is to advance ideas for energy-efficient electronic hardware for artificial intelligence and to facilitate their development. At the start of the competition in September 2019 an ECG dataset was made available to train a classification model. At the end of the competition in December 2020, a previously unknown test dataset consisting of 500 additional ECG recordings has to be classified. Each submission must detect at least 90 % of patients with atrial fibrillation in less than 5 minutes with a maximum of 20 % false positives. The method used has to be based on machine learning and should be sufficiently general to be adaptable to completely different tasks without conceptual changes. If all these conditions are met, the ranking in the competition is based on the total energy consumption of the system during classification. Any initialization and loading of the test data into an internal memory is not recorded; the measurement of energy consumption only includes the actual inference from the raw data to one of the two classes.

The classification of the datasets is planned with convolutional neural networks, which will be trained directly with the neuromorphic BrainScaleS-2 system. As part of the competition, the working group is developing a new extension for the machine learning framework PyTorch to transparently integrate the hardware. In the context of this thesis this is extended and equipped with an additional simulation mode to allow experiments without access to the BrainScaleS-2 hardware. To achieve good results with the limited resolution of the hardware it will be necessary to develop a well-dosed preprocessing of the datasets, which is adapted to the requirements of the analog system.

2 Fundamentals

This chapter covers the background of the applied machine learning algorithms, the use of the neuromorphic application specific integrated circuit BrainScaleS-2 as an accelerator for analog inference and its planned integration into the machine learning framework PyTorch. This is followed by an introduction to the datasets used, i. e., electrocardiogram recordings and smartphone sensor data, and a description of their important features.

2.1 Basics and Terminology of Machine Learning

The goal of machine learning is to enable computers to solve tasks without having to program them explicitly. Machine learning algorithms attempt to learn arbitrary input-output relations typically based on a huge number of training examples. With this so-called model the computer is then able to make predictions or decisions without any man-made rule. A key aspect of machine learning is ensuring a high degree of generalization, thereby enabling the model to perform well even on previously unseen examples. This section covers the basic concepts and terminology, for detailed standard works I refer to Hastie, Tibshirani, and Friedman (2009) and Bishop (2007).

2.1.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are an approach to adapt the structure of signal processing in a biological brain. This is achieved by a directed graph-like structure, as sketched in figure 2.1. The nodes, which are typically arranged in layers, are inspired by biological neurons and loosely model their biological equivalent. In this analogy, the weighted edges correspond to the synapses of their biological counterpart, as they transmit the signal from one neuron to the other. In contrast to, e. g., spiking models, which more closely mimic natural neural networks, the neurons in an artificial neural network usually have no internal dynamics, they only represent an input-output relation.

The output of a node in an ANN is the weighted sum of its inputs, on which a non-linearity φ is applied afterwards. This transfer function φ forms the activation function as it models the neuron activity, i. e., the output response of the neuron to its membrane potential u_k. For the k^{th} node of the l^{th} layer with n_{in} inputs $x_j^{(l-1)}$ from the preceding layer and the respective weights $w_{jk}^{(l)}$, this results in

$$x_k^{(l)} = \varphi\left(u_k^{(l)}\right) \quad \text{with} \quad u_k^{(l)} = \sum_{j=1}^{n_{\text{in}}} x_j^{(l-1)} w_{jk}^{(l)} + b_k^{(l)} \,. \tag{2.1}$$

The bias b_k adds an additional offset and enables to adjust the sensitivity of the neuron. Until a few years ago, a sigmoid was commonly used as an activation function in neural

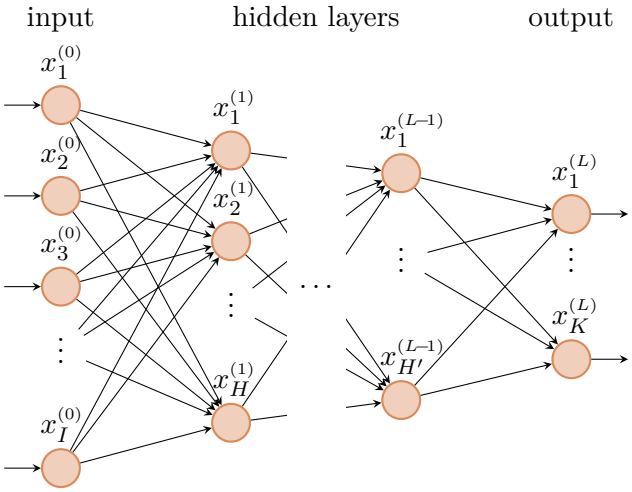

Figure 2.1: Schematic of a densely-connected multi-layer feed-forward artificial neural network with L layers, I inputs and K output units. The neurons of successive layers are connected by the weights w_{ij}.

networks (Hastie, Tibshirani, and Friedman, 2009), which is defined as

$$\varphi^{\text{sigmoid}}(u_k) = \frac{1}{1 + \exp(-u_k)} \,. \tag{2.2}$$

It is symmetric around zero and also forms the basis for biologically inspired rate-based models of the brain. Nowadays, the rectified linear unit (ReLU) is used in most cases, as it is easier to compute and similarly effective. It is defined as

$$\varphi^{\text{ReLU}}(u_k) = \max(0, u_k) \,. \tag{2.3}$$

An artificial neural network typically consists of multiple successive layers. Its input layer differs from the subsequent ones in that its value is not the result of the calculation in equation (2.1), but represents the vector of the input sample $\left\{x_k^{(0)}\right\}$. Such a feed-forward network is very powerful for a lot of different tasks. It was shown by Leshno et al. (1993) that with a sufficiently large number of hidden neurons any continuous function can be represented. An artificial neural network with more than one hidden layer between input and output layers is often referred to as deep neural network (DNN).

Neural networks are widely applicable and suitable for both regression and classification problems, the underlying structure remains the same. For regression using a L-layer network there is typically only one output unit $x_0^{(L)}$ in the last layer, but also multi-dimensional relations can be represented by such a network. For a classification with K classes the network has one output for each class, where the output value is a measure for the probability of the respective class.

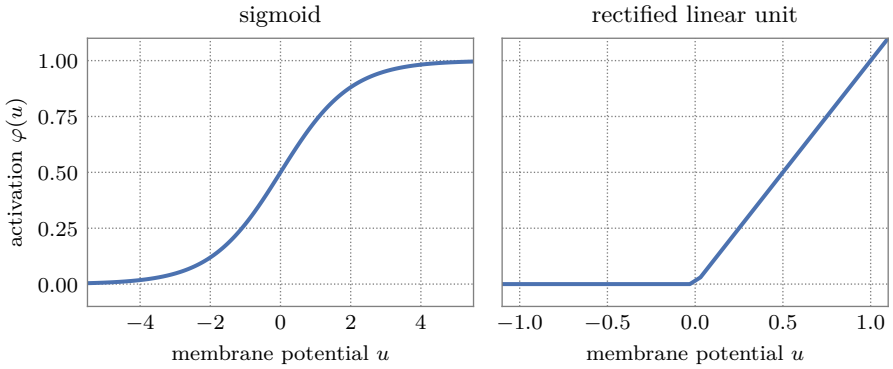

Figure 2.2: Frequently used activation functions. In contrast to the sigmoid (left) the recified linear unit (right) is not symmetric around zero.

2.1.2 Supervised Learning

Supervised learning aims to learn the relationship between an input and a given output in a dataset. Such a dataset consists of M input output pairs (x_i, y_i). It is assumed that there exists a relation F that can be approximated by an artificial neural network such that

$$y_i = F(x_i) \tag{2.4}$$

holds. The idea is that the approximated model can subsequently be applied to unknown inputs x_i' to get a predicted result $F(x_i')$.

To obtain a measure of the performance of such a model, a so-called loss function is used. For regression, i. e., if F is a continuous function, the L2 loss is often a reasonable choice, using the mean-square deviation of the predicted to the true value

$$\mathcal{L}^{\mathrm{L2}} = \frac{1}{M} \sum_{i=1}^{M} ||y_i - F(x_i)||^2 \,. \tag{2.5}$$

Here the sum goes over the complete dataset of length M, but it can also be approximated by taking only parts of the data.

Now, the goal during the supervised training is to minimize the loss function to increase the accuracy of the network. In the simplest case, optimization can be accomplished via gradient descent with a learning rate η as parameter. The weights w_{jk} of the network are updated iteratively using

$$w_{jk}^{(l)} \to w_{jk}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}} \,. \tag{2.6}$$

For a sufficiently small learning rate this converges to a local minimum of the loss function $\mathcal{L}$. Since the partial derivative contains a sum over the whole dataset it is quite expensive to calculate. Therefore, it is often approximated by taking only a part of the dataset or even only a single sample, which is randomly selected in each iteration. This procedure is

referred to as stochastic gradient descent or mini-batch learning (Masters and Luschi, 2018).

Another typical example of supervised learning are classification tasks: an algorithm is supposed to separate different classes of data, e. g., objects in images or an earthquake from different sensor inputs. For an ANN as shown in figure 2.1, the resulting predicted class $\hat{y}$ corresponds to the arg max of the output from the last network layer, i. e.,

$$\hat{y} = \underset{k=1,\dots,K}{\arg\max} \left\{ x_k^{(L)} \right\} . \tag{2.7}$$

The most common loss function for classification problems is the cross-entropy loss, which is defined as the negative log-likelihood function. To get the probability p_k for the class k, the softmax function is applied to the vector of the raw non-normalized outputs y_k from the last layer of the classification model, i. e.,

$$p_k = \frac{e^{y_k}}{\sum_i \mathrm{e}^{y_i}} . \tag{2.8}$$

It provides a vector of normalized probabilities with one value for each of the possible classes. The cross-entropy loss over the whole dataset is then defined as

$$\mathcal{L} = -\frac{1}{M} \sum_{m=1}^{M} \sum_{k=1}^{K} (y_m)_k \cdot \log(p_k) . \tag{2.9}$$

As before, M denotes the number of elements in the training data set and K the number of units of the last layer.

2.1.2.1 Back-Propagation

Back-propagation is the application of the chain rule of derivatives and is used to calculate the partial derivative of each layer according to each weight in a network as defined in equation (2.1). For this purpose the derivatives of the deeper layers can be reused in calculating error signals, i. e., partial derivatives, for the earlier layers. As derivative propagates from the output back to the input, this is referred to as back-propagation (Rumelhart, Hinton, and R. J. Williams, 1986). Nevertheless, the calculation of the individual gradients is still complex due to a lot of vector-matrix multiplications, which is why machine learning started to become popular with the recent availability of powerful computing units.

The back-propagation formula can be derived by applying the chain rule to the derivative term. For a single sample one gets the recursive formula

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial x_k} \cdot \delta_{kj}^{(l)} \cdot x_i^{(l-1)} \tag{2.10}$$

$$\text{with} \qquad \delta_{ij}^{(l)} = \varphi'\left(u_i^{(l)}\right) \cdot w_{\gamma i}^{(l)} \cdot \delta_{\gamma j}^{(l+1)} . \tag{2.11}$$

The result can be used in conjunction with equation (2.6) to calculate weight updates during training. Machine learning frameworks, like Tensorflow (Martín Abadi et al., 2015) or PyTorch (Paszke et al., 2019), are specialized to follow the path of the data through the applied operations and automatically calculate the corresponding gradients.

2.1.2.2 Issues in Training Neural Networks

Supervised learning is an illustrative case for some issues in training artificial neural networks. A difficult task, for example, is the choice of the hyperparameters of the model, such as the number of neurons in the hidden layers, cf. figure 2.3. If the model is not powerful enough to adapt to the relation between input and target, one talks of underfitting. Such a high-bias model achieves equally poor results on the training and test data, even continuing training for any length of time does not lead to better results. Overfitting occurs when the model adapts too much to the noise of the training data. Although the targets are reached quite well during the training, the model is not able to adapt the underlying relation. Thus the loss on the training set becomes very small, but the loss on a separate test dataset not presented during training is much higher.

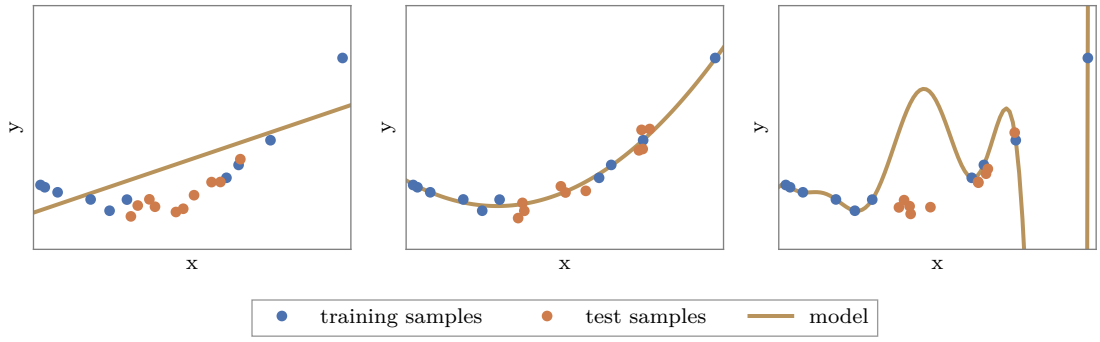

Figure 2.3: Example of over- and underfitting. Left: a linear model is unable to represent the dataset, this is known as underfitting. The model similarly fails on both the training set and the test set. Center: the dataset can best be described with a second order polynomial. The average loss on test and training data is about the same. Right: fitting a higher-order polynomial reduces the loss on the training set by adapting to its noise. However, this model is not able to predict the test samples.

The loss function $\mathcal{L}$ is not convex in general, i. e., it usually possesses many local minima. Therefore, the choice of the initial weights has a decisive influence on the result of training. Different methods for initializing the weights (cf. section 2.1.4) can have a significant impact on the training success. Retrying with a different set of random initial weights can improve the performance of the trained model, too.

2.1.3 Layer Architectures

Figure 2.1 shows one-dimensional, fully-connected layers. In general, layers can also be multi-dimensional and have different shapes. However, variants without inter-layer connections can be derived from the one-dimensional case.

This is done by enumerating the units of the layer and arranging them in the same vector, possibly with an additional duplication of weights. Nevertheless, the basic principle

of multiplying vectors with weight matrices and then applying an activation function remains the same. Today there are a number of more advanced layer types that break with the original scheme of a feed-forward network by adding additional features like inter-layer connections. A prominent example are long short-term memory (LSTM) layers proposed by Hochreiter and Schmidhuber (1997). For acceleration on the mixed-signal neuromorphic BrainScaleS-2 chip, however, at first only the stateless components of feed-forward artificial neuronal networks will be supported since their data flow is easier to implement.

Linear Layers Layers that are fully-connected, i. e., every neuron is connected to the output of every neuron of the preceding layer are referred to as *dense* or *linear* layers. They are the classical components of a multi-layer perceptron as introduced in figure 2.1. A decisive disadvantage of this kind of layer is the high number of parameters to be learned and stored, it scales linearly with the number of inputs. Therefore dense layers are mostly used for final layers at the end of a model, where the number of neurons is typically much smaller.

Convolutional Layers Convolutional layers operate on data that are spatially or temporally arranged. This typically includes image information (2 dimensions), but also 3-dimensional data like videos, or 1-dimensional data like audio files or sensors that provide scalar quantities over time. In addition, the data has often so-called channels, for example color information in images and videos, left and right signals of an audio recording or different leads in an electrocardiogram.

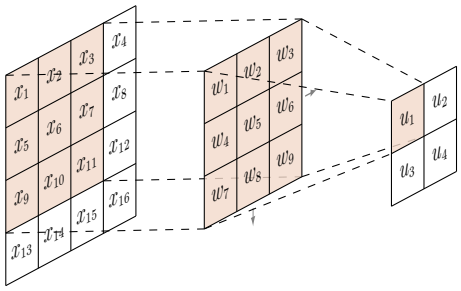

Figure 2.4: Example of a discrete convolution operation. A 3×3 filter kernel w_j is strided in both directions over a 4×4 input x_i to form the output u_k. The active regions to compute the output value are shaded orange. For an input with multiple channels the same number of filters is needed, which themself can have an additional output dimension. This way the output may also include several channels.

In convolutional layers, the input image is convolved with one or multiple filter matrices that are typically small in comparison to the dimensions of the input, cf. figure 2.4. This filter takes a slice of the input with all its channels and examines it for a certain property, such as a certain shape or composition of colors, as can be seen in figure 2.5. The output

of a single neuron is then a measure of how well a single filter fits to the image at exactly this point. This way, different details are identified along with their location of occurrence. Since not all neurons in two consecutive layers are connected to each other and a lot of the neurons share the same weights, such a convolution needs much less parameters than a fully-connected layer. A great advantage of this type of layer is also the local independence of the filters. If similar features occur in different positions, the same filter is still trained and used for inference. This reduces the training time and a smaller number of training examples may be sufficient.

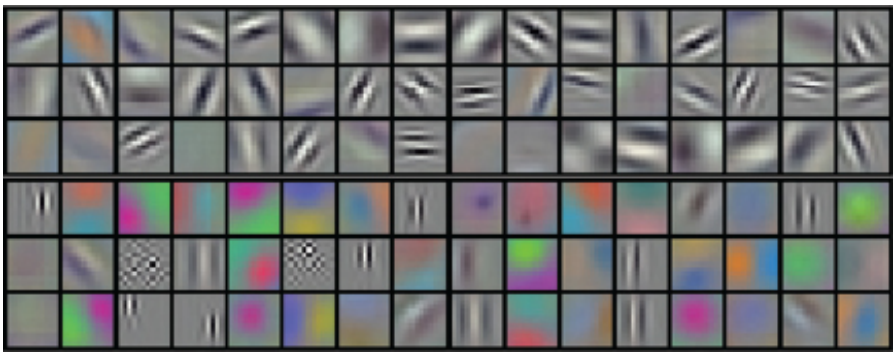

Figure 2.5: 96 convolutional $11 \times 11 \times 3$ kernels learned by the first convolutional layer of a model proposed by Krizhevsky, Sutskever, and Hinton (2012). This first application of a rather deep convolutional neural network is known as AlexNet. In 2012 it achieved breakthrough performance on Google's ImageNet challenge (Deng et al., 2009) that aims at recognizing objects in $224 \times 224 \times 3$ images. The kernels give an idea of how this was achieved: the model specifically looks for the orientation of edges and colors. In the following layers the information where a certain pattern occurs is further processed in order to recognize the object in the image.

A deep neural network that contains one or more convolutional layers is referred to as convolutional neural network (CNN). A typical CNN often consists of alternating convolutional and pooling layers followed by a small number of fully-connected layers. The pooling aims at subsequently reduce the size of the input data for the next layer. This is done by combining a number of directly adjacent points to one point by, e. g., using their maximum. Max-pooling adds another non-linearity which might be advantageous as it sharpens the view of the network. However, it has been shown by Springenberg et al. (2014), that in common image recognition networks max-pooling can be replaced by a convolutional layer with additional stride without diminishing the accuracy.

2.1.4 Initialization

For small artificial neural networks with only a few layers, the choice of the initial weights does not make a significant difference. Therefore, in the history of such networks the weights were often drawn from random distributions with fixed properties. Various

standard values have been established for this, for example Hastie, Tibshirani, and Friedman (2009) propose random uniform weights over a range from -0.7 to 0.7 as a typical choice for standardized inputs with mean zero and standard deviation one.

However, with the recent increase of networks complexity and their number of successive layers, this approach turned out to be unsuitable. It was observed that in some deep neural networks this kind of initialization leads to exponentially increasing activations, which may even exceed the available value range, whereas in other networks the activations nearly vanish. This can significantly slow down or even completely prevent the convergence of the model during training.

2.1.4.1 Xavier Initialization

A solution for this problem was proposed by Glorot and Bengio (2010). The underlying idea is to keep the variance of the signal on the forward and the backward pass as constant as possible throughout the network, thus avoiding runaway effects.

The neurons output activation and therefore the input for the subsequent layer has been defined in equation (2.1). Simplifying the activation function as $\varphi(\boldsymbol{u}) = \boldsymbol{u}$ first, the equation becomes

$$\boldsymbol{u}^{(l)} = W^{(l)} \cdot \boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)} \; . \tag{2.12}$$

Further it is assumed that the entries in $\boldsymbol{u}$ and W are independently and identically distributed, and u, x and w represent the corresponding random variables of their entries. For the forward pass this leads to

$$\mathrm{Var}[u^{(l)}] = n_{\mathrm{in}}^{(l)} \cdot \mathrm{Var}[w^{(l)} \cdot x^{(l)}] \; . \tag{2.13}$$

Here $n_{\mathrm{in}}^{(l)}$ denotes the fan-in, i. e., the number of non-vanishing inputs to the l^{th} layer. If both, w and x, have zero mean, this simplifies further to

$$\mathrm{Var}[u^{(l)}] = n_{\mathrm{in}}^{(l)} \cdot \mathrm{Var}[w^{(l)}] \cdot \mathrm{Var}[x^{(l)}] \; . \tag{2.14}$$

Following these considerations, the output of the whole model with L layers has a variance of

$$\mathrm{Var}[u^{(L)}] = \mathrm{Var}[x^{(0)}] \cdot \prod_{l=1}^{L} n_{\mathrm{in}}^{(l)} \cdot \mathrm{Var}[w^{(l)}] \; . \tag{2.15}$$

This shows again how the choice of the initial weights has a decisive influence on the model performance: the product in equation (2.15) and thus the variance of the output for large L diverges with improper choice of weights. In order to avoid reducing or magnifying the magnitudes of the input signals exponentially, the random variable of the weights has to fulfill

$$n_{\mathrm{in}}^{(l)} \cdot \mathrm{Var}[w^{(l)}] = 1, \quad \forall l \; . \tag{2.16}$$

In terms of the standard deviation of a zero-mean Gaussian distribution $N(0, \sigma)$ and the boundary of an equivalent uniform distribution $\mathcal{U}(-r, r)$, this corresponds to

$$\sigma^{(l)} = \sqrt{\frac{1}{n_{\mathrm{in}}^{(l)}}} \qquad \text{and} \qquad r^{(l)} = \sqrt{\frac{3}{n_{\mathrm{in}}^{(l)}}} \; , \tag{2.17}$$

respectively. This method of initialization is applied by the machine learning framework Tensorflow (Martín Abadi et al., 2015) by default, whereby a uniform distribution is used.

2.1.4.2 Kaiming Initialization

Another method for the initialization of deep neural networks, which achieves even better results with recent deep neural network models, was presented by He et al. (2015). This method is based on the key ideas of the Xavier initialization already introduced in section 2.1.4.1, but adds an additional gain factor g^{φ}, that depends on the activation function φ.

For the derivation of equation (2.14) it is assumed that the input of each layer has zero mean. However, this only holds for symmetric activation functions, e. g., a sigmoid, as the output of each layer also forms the input to the subsequent layer, i. e., $\boldsymbol{x}^{(l+1)} = \varphi(\boldsymbol{u}^{(l)})$. Because of this, equation (2.13) results in

$$\mathrm{Var}[u^{(l)}] = n_{\mathrm{in}}^{(l)} \cdot \mathrm{Var}[w^{(l)}] \cdot \mathrm{E}[(x^{(l)})^2] \tag{2.18}$$

as in general $\mathrm{Var}[x] \neq \mathrm{E}[x^2]$ for the expectation E of the square of x. Nevertheless, to consider the variances, an additional factor is proposed by He et al. (2015) such that

$$\mathrm{E}[x^2] = g_{\varphi}^2 \cdot \mathrm{Var}[x] \tag{2.19}$$

applies. If the activation φ is a ReLU, the gain factor becomes $g_{\varphi} = \sqrt{2}$, as the ReLU cuts the parameter space in half by clipping negative numbers (cf. figure 2.2).

The parameters of a suitable Gaussian or uniform distribution for the initial weights can be derived with the same considerations as in section 2.1.4.1 resulting in

$$\sigma^{(l)} = g_{\varphi} \cdot \sqrt{\frac{1}{n_{\mathrm{in}}^{(l)}}} \qquad \text{and} \qquad r^{(l)} = g_{\varphi} \cdot \sqrt{\frac{3}{n_{\mathrm{in}}^{(l)}}}\,. \tag{2.20}$$

This method is applied by default by PyTorch (Paszke et al., 2019) when initializing the weights, whereby an uniform distribution is used and a ReLU is assumed as activation function.

2.2 The BrainScaleS-2 Neuromorphic Hardware

The BrainScaleS-2 chip is the most recent version of a series of neuromorphic application-specific integrated circuits (ASICs). It is a very versatile experiment tool designed for the accelerated emulation of synapses, neurons and plasticity models in spiking neuronal networks (Schemmel, Billaudelle, et al., 2020). The aim is to provide a realistic emulation of the brain's components and to enable studies of the principles and dynamics of the brain. In contrast to other systems, it is a real physical emulation of the components of the brain, manufactured in standard 65 nm CMOS technology.

The analog core of BrainScaleS-2 consists of 512 silicon neurons, connected to 256 synapses, each. As might be concluded from the right picture in figure 2.6, the

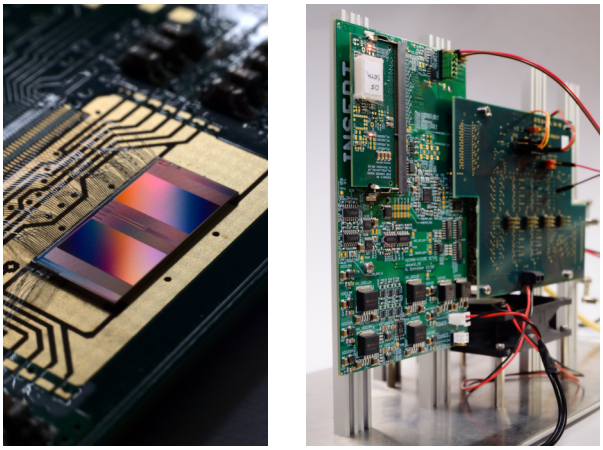

Figure 2.6: On the left: the bonded BrainScaleS-2 neuromorphic ASIC. Right: hardware setup including power-, communication- and FPGA-boards. The actual chip is covered by the white cap in the upper left.

BrainScaleS-2 ASIC is spatially divided into two almost identical parts. Each half of the chip comes with a 256×256 synapse array and 256 connected neurons. The analog core of synapse and neuron circuits is accompanied by two digital general-purpose SIMD processors (plasticity processing units; PPUs) at the two ends on the same substrate. These embedded specialized micro-controllers are designed to handle the update rules of the networks and to control experiments autonomously.

The current hardware setup is shown on the right side of figure 2.6. It provides the power supply components as well as test connections. It also features a field programmable gate array (FPGA) that handles the communication to the outside world. This includes the integration of memory and a network interface. While FPGA and PPU together are capable of running standalone experiments, a host computer is used for this purpose throughout this thesis that is connected to the hardware setup via Ethernet. This allows the use of conventional tools like PyTorch, which cannot be compiled for the PPU, to handle the training and repeatedly compute and adjust updated experiment parameters.

2.2.1 Matrix Multiplication on BrainScaleS-2

Due to its analog nature and its versatile configurability, the BrainScaleS-2 ASIC can also be used for energy-efficient acceleration of conventional machine learning applications. A more detailed description on how this is implemented can be found in Weis et al. (2020).

The value of an input is multiplied by the weight of a synapse and the result flows as an electric current to a neuron. The input encodes the time for which a current flows onto the neuron membrane, the value of the synapse weight corresponds to its amplitude. The resulting charge at the membrane of the neuron is accumulated with the results of further multiplications in the same column. The result of this multiply accumulate operation (MAC) is then proportional to the total charge collected by the neuron and

thus to the electric potential over the membrane of the neuron. This voltage is translated back into a digital value by an analog-to-digital converter (ADC). Although the ADC has a nominal resolution of 8 bit, the expected resolution of the entire operation is lower due to fluctuations resulting from the analog nature of the hardware.

	Max. size	Resolution	Value range [LSB[1]]
input	128	5 bit	$0 \dots 31$
weight	128×256	(6+1) bit	$-63 \dots 63$
output	256	8 bit	$-128 \dots 127$

Table 2.1: Available parameter properties for a typical configuration of BrainScaleS-2 when using the multiplication mode. All values are integers. The *input* encodes the time in which a current flows onto the neuron membrane in 5 bit resolution. Since negative time intervals are not possible, only positive values are allowed. The *weight* determines the current that flows to the neuron membrane, its resolution is 6 bit plus sign. The *output* refers to the neuron membrane potential, which is digitized with 8 bit accuracy. However, the result is only approximately linear between −100 LSB and 100 LSB, saturation may occur for certain neurons above and below this range and the operation becomes quite non-linear.

In a first linear approximation it can be assumed that the operation on the chip can be represented by

$$y_j = \sum_i w_{ij} \cdot x_i \cdot g_{\text{BSS-2}} + \kappa_j \tag{2.21}$$

The input vector $\boldsymbol{x}$, the weight matrix W and the output $\boldsymbol{y}$ are all integers within different ranges, which are summarized in table 2.1. The factor $g_{\text{BSS-2}}$ depends mainly on the membrane capacitance of the neurons, the time constant of the synaptic inputs and the leakage currents and is approximately in the range from 0.001 to 0.005. The noise $\boldsymbol{\kappa}$ can well be assumed to be Gaussian distributed with a standard deviation of about 2 LSB.

However, this operation is subject to some peculiarities that can make its use in neural networks considerably more difficult. Due to the analog nature of the chip, for example, it is not irrelevant whether high values in the multiplied vector are directly adjacent to each other or more distant, because too high currents in short time may saturate the synaptic input. Furthermore the operation is not perfectly commutative due to nonlinear effects, e. g., $10 \cdot 20 \neq 20 \cdot 10$ in general.

Multiple Operations in Parallel With an additional label bit at the inputs it is possible to send up to two vectors in parallel to the synapse array. Each individual synapse is then sensitive to one of the two values in its row, depending on its encoding. By this mechanism, for example, two smaller matrices can be written side by side on

[1] unit that corresponds to the smallest step that can be represented by the hardware

the same synapse array and simultaneously multiplied by different input vectors. It also allows to split up large weight matrices and place the individual parts next to each other.

2.2.2 PyTorch Integration

The PyTorch (Paszke et al., 2019) extension `hxtorch` is planned as one of two high level APIs besides `pynn-brainscales` (Czierlinski, 2020), an extension to PyNN (Davison et al., 2009), which forms an abstraction layer for the emulation of spiking neurons and synapses on BrainScaleS-2. The goal of `hxtorch` is a transparent integration of the multiply accumulate operation described in section 2.2.1. Parts of this have been already implemented by Eric Müller, Philipp Spilger and Christian Pehle, especially the integration of the MAC operation which is executed on the ASIC as indicated above. A graph-based backend that `hxtorch` uses to perform this task is subject of a further master's project by Philipp Spilger and will be explained in detail in his upcoming thesis.

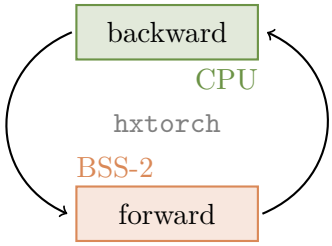

Figure 2.7: The principle of training with hardware in the loop using `hxtorch`. The forward path is executed on the BrainScaleS-2 hardware. The gradients in the backward path are calculated with a software model on the host computer.

Since the chip is not able to calculate the gradient of the MAC operation, it is planned to perform this task on a host computer during training, as sketched in figure 2.7. This scheme was already implemented before on BrainScaleS for spiking neuronal networks by Schmitt et al. (2017) and Cramer et al. (2020), the latter also uses PyTorch for calculating and handling of the gradients and weight updates.

2.3 Datasets

The datasets used in this thesis are presented in detail on the following pages. In each case they are one-dimensional, time-based sensor data. A significant advantage of this type of data is the low bandwidth compared to, e. g., videos. Furthermore, the artificial neural networks required for classification are much smaller compared to other problems like object recognition in images and videos. Therefore, they are well suited to study the basic principles without the need for too much memory and computing power. This makes them also ideal for embedded applications.

2.3.1 Electrocardiography

Measuring the electrical activity of the heart using surface electrodes on the skin is known as electrocardiography. The resulting electrocardiogram (ECG) is a set of graphs representing the temporal course of electrical potential difference between two points, each. The observable voltages are caused by the electrical conduction system of the heart, which coordinates the complex sequence of muscle contractions to pump blood through the body. The heart rate can be easily estimated by counting the distinct heart beats in a certain interval. According to the American Heart Association, the normal resting heart rate of an adult human should be in the range from $60\,\mathrm{min}^{-1}$ to $100\,\mathrm{min}^{-1}$, which corresponds to an average interval of 0.6 s to 1.0 s. However, there is an additional number of features in an ECG, which are affected by cardiovascular diseases and abnormalities.

2.3.1.1 ECG Features

Some of the features that can be obtained from an ECG are shown in figure 2.9. Unless stated otherwise, all numbers in this paragraph are taken from Rangayyan (2002).

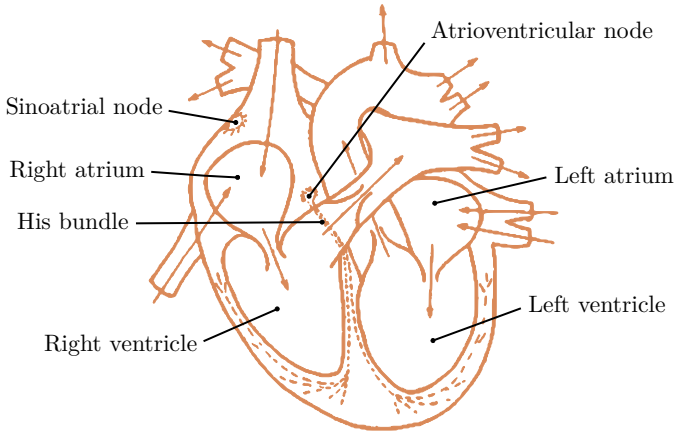

Figure 2.8: Schematic representation of the human heart including the conduction system. In a healthy heart, the electrical signal originates at the sinoatrial node, is conducted through the atria and passed via the atrioventricular node to the ventricles, where it causes a heartbeat. Adapted from Rangayyan (2002).

In a healthy heart, the electrical signal originates at the sinoatrial node located in its right atrium. The signal propagates through the atria via gap junctions, as they are also found in electrical synapses in the brain, causing first the right and then the left atrium to contract. This can be observed on an ECG as P wave (cf. figure 2.9). It is a slow, low-amplitude wave with a duration of about 80 ms and an absolute amplitude of about 0.1 mV to 0.2 mV. It is much smaller than most of the other features, depending on the spatial orientation of the heart and the leads used, it may be hard to recognize.

On its way through the heart the signal then passes the atrioventricular node (AV node), which electrically connects atria and ventricles. Hereby the signal is delayed by

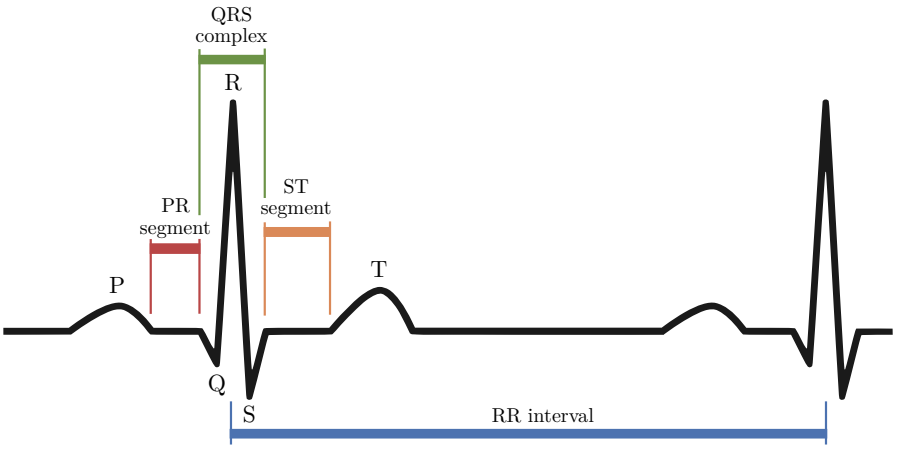

Figure 2.9: The individual components of the electrocardiogram of a simplified normal sinus rhythm signal. The QRS complex, including the R peak, is the most prominent and thus defines the beat position. The actual manifestation of the graph depends on individual factors, such as the position and orientation of the heart in the thorax, the physiological characteristics of the components of the heart and the placement of the electrodes on the skin. Missing or altered features can indicate an anomaly and thus a heart disease. In particular, atrial fibrillation can be diagnosed by irregular RR intervals and missing P wave.

about 60 ms to 80 ms, which is referred to as *PR* segment. An important property of the atrioventricular node is its refractory period, which protects the ventricles from excessively fast contractions. In addition, the atrioventricular node possesses a slow intrinsic firing rate to ensure a minimum supply of blood in case the electrical conduction system of the atria breaks down. This mechanisms ensure that the frequency of the heart beats remains within a range of about $40\,\mathrm{min}^{-1}$ to $300\,\mathrm{min}^{-1}$ (Guyton and Hall, 2006).

The following contraction of the ventricles is the main component of the heartbeat. A rapid depolarization produces a sharp *QRS* complex with an amplitude of about 1 mV and a duration of 80 ms. This part is the most pronounced, which is why the *R* peak is often used to define the beat position.

The action potential of the ventricular muscle cells is of relatively long duration and typically lasts 300 ms to 350 ms. Thereby an iso-electric plateau of 100 ms to 120 ms after the *QRS* complex, the *ST* segment, can be observed first. The subsequent diffuse relaxation of the muscle cells then causes the slow *T* wave with a duration of 100 ms to 160 ms and an absolute amplitude of 0.1 mV to 0.3 mV.

2.3.1.2 ECG Signal Acquisition

A pair of electrodes attached to the body surface forms a lead. An electrocardiogram usually consists of several channels; the standard 12-channel ECG uses three limb electrodes and six additional electrodes around the chest. The 12 leads are calculated from potential differences between these points. However, especially in mobile applications, only

three (sometimes even only two) limb electrodes are used, cf. figure 2.10. This limits the observation to the amplitude and orientation of the signal projected on a two-dimensional vertical plane through the body. The most common leads are *I*, *II* and *III*, based on electrodes at the right and left arm, and left leg. From these the so-called augmented leads *aVR*, *aVL* and *aVF* can be derived, which have their reference point in the middle of the body, shown in the lower part of figure 2.10. Reversing the arm electrodes causes lead *II* and *III* to switch and a changed polarity of lead *I*.

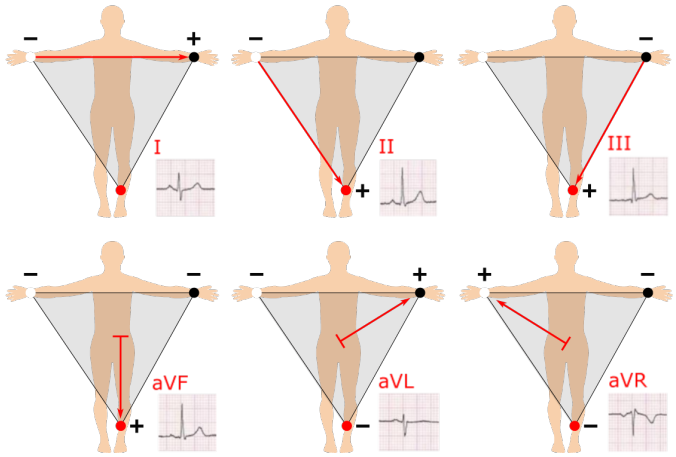

Figure 2.10: The limb leads, also known as Einthoven's triangle. The leads *I*, *II* and *III* in the upper part represent actual differences between two of the electrodes at the arms and the left leg. The augmented leads *aVR*, *aVL* and *aVF* in the lower half of the figure are each derived from all potentials and have their reference point in the center of the body. Adapted from Patchett (2015).

2.3.1.3 Challenges

There are a number of challenges that one encounters when trying to process ECG data, and a lot of them apply to other sensor data as well. These include noise, baseline wandering, changing signal amplitudes and shape abnormalities. Common noise sources are 50 Hz or 60 Hz power line interference, other muscle activity and movements of the electrodes on the patient's skin.

An aggravating factor in ECG recordings is that the actual manifestation of the graph depends on many individual factors. These include the exact position and orientation of the heart in the thorax, the physiological characteristics of the various components of the heart in relation to each other and the placement of the electrodes on the skin. Therefore only few signals look like sketched in figure 2.9, certain details can appear weaker, inverted or may even be hidden. An illustrative example of this is shown in figure 2.12, where the first three lines correspond to normal sinus heart rhythms and yet have very different shapes. This makes the interpretation of an electrocardiogram challenging even for experienced cardiologists.

2.3.1.4 Atrial Fibrillation

Atrial fibrillation is the most common serious abnormal heart rhythm and affects more than 33 million people worldwide (Chung et al., 2020). It is defined as a "supraventricular tachyarrhythmia characterized by uncoordinated atrial activation with consequent deterioration of atrial mechanical function" by the American College of Cardiology (ACC), the American Heart Association (AHA) and the European Society of Cardiology (ESC) in Fuster et al. (2006). This means that the electrical conduction system of the heart is disordered. The electrical signal, that stimulates the ventricles to contract, no longer originates from the sinoatrial node. Instead, the signal propagates continuously and chaotically through the atria. Whenever the electrical signal passes the atrioventricular node and the node is excitable, it enters the ventricles and triggers a contraction. This results in a replacement of the P waves (cf. figure 2.9) by rapid oscillations or fibrillatory waves (f waves) and causes an irregular heart rhythm. It can be observed for example in tape 201 in figure 2.12, where such oscillations and irregular beats occur.

2.3.1.5 The BMBF Competition Dataset

As part of the competition "Energieeffizientes KI-System" (German; *Energy-efficient AI system*) of the Federal Ministry of Education and Research (BMBF, 2019) a training dataset was made available. Its composition is listed in table 2.2. It contains 16000 two-channel electrocardiogram recordings, half of them with normal sinus rhythm and half with atrial fibrillation. They are sampled at 512 Hz and are about 120 s in length.

Type	Num of recordings	Time length [s]		
		Min	Max	Mean
Normal	8000	119.26	120.26	119.78
AFib	8000	119.23	120.26	119.78
Total	16000	119.23	120.26	119.78

Table 2.2: Composition of the training dataset of the BMBF competition.

The original data were taken from the *Telemedical Interventional Monitoring in Heart Failure* study (TIM-HF) conducted at the Charité in Berlin (Koehler et al., 2011). The ECG data were most likely recorded by the subjects themselves at home with a portable heart monitor. It is quite possible that individual electrodes were swapped and the channels in the data may sometimes be inverted or interchanged. As the dataset is subject to a non-disclosure agreement, this thesis does not contain any raw data, but only the results on this dataset. However, figure 2.11 shows examples for the contained classes from a very similar dataset.

Each ECG recording is stored to a separate file, which is made up of a text header terminated by a `\0` character, followed by a binary part. This part contains the individual data points, alternating for each of the two channels, as 16 bit unsigned integers, where only 12 bit are used. The header contains information about an offset β and factor γ to

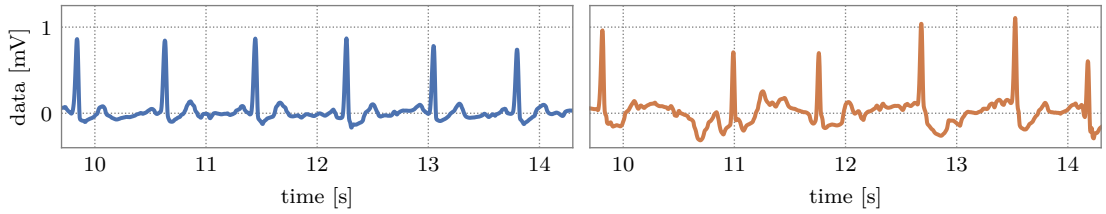

Figure 2.11: Examples for the two classes to distinguish in the BMBF competition: normal sinus rhythm (left) and atrial fibrillation (right). Since the provided ECG recordings are subject to a non-disclosure agreement, they are not published in this thesis. Instead, these examples were taken from the *PhysioNet Challenge 2017* (Clifford et al., 2017).

convert these values y_{raw} to potential differences in mV:

$$y(t)\ [\text{mV}] = (y_{\text{raw}}(t) - \beta) \cdot \gamma\ . \tag{2.22}$$

With $\beta = 2048$ and $\gamma = 0.002\,93\,\text{mV}$, this results in a covered range from $-6\,\text{mV}$ to $6\,\text{mV}$.

Many of the data contain very pronounced artifacts, probably due to additional muscle contractions during the measurement. Here, the signal alternates rapidly between the two extreme values, whereby it even saturates, hiding the characteristic parts of the heartbeat. This seems to be particularly pronounced within the first 10 s of the trace and becomes less frequent afterwards.

2.3.1.6 PhysioNet Challenge 2017

A very similar task as the BMBF competition was subject of the *PhysioNet Challenge* in 2017 (Clifford et al., 2017). It aimed to encourage the development of algorithms for ECG classification. The challenge was to predict whether ECG recordings show normal sinus rhythm, atrial fibrillation, an alternative rhythm, or are too noisy to be classified. Unlike the BMBF challenge, energy consumption and suitability for embedded devices did not play a role here.

Type	Num of recordings	Time length [s] Min	Max	Mean
Normal	5154	9.0	61.0	31.9
AFib	771	10.0	60.0	31.6
Other	2557	9.1	60.9	34.1
Noisy	46	10.2	60.0	27.1
Total	8528	9.0	61.0	32.5

Table 2.3: Composition of the training dataset of the PhysioNet Challenge 2017.

The dataset provided contains 8528 ECG single-lead recordings from 9 s to 61 s in length, sampled at 300 Hz. Since the data of the examples with atrial fibrillation are, as

with the BMBF dataset, arrhythmic over their entire length and do not contain only a few individual periods, they are ideally suited as reference data for testing the presented models.

2.3.1.7 The MIT-BIH Databases

Since 1975, laboratories at Boston's Beth Israel Hospital and at MIT conducted research on arrhythmia analysis. In the course of this, they created a database of ECG recordings which they distributed from 1980 onward, first on analog tapes and from 1989 on CD-ROM. This was the first generally available set of standard test material for evaluation of arrhythmia detectors, and is therefore very often used in scientific publications, which makes them a reference for arrhythmia analysis. The database contains 30 min excerpts of 48 recordings out of a total of 4000, some of which were randomly selected, enriched with less common but important arrhythmias, Accompanying, about 110000 annotations are available, which were created by at least two independent cardiologists and contain the positions of the heartbeats and their type. Thereby a differentiation is made between 18 different beat shapes. But also interesting areas like the beginning and end of an arrhythmic episode or noise are labeled. Figure 2.12 shows an excerpt of the contained samples with annotated beat positions. It is remarkable that the recordings strongly differ in shapes and amplitudes. This first published database is named **MITDB**.

In the meantime, the collection of MIT-BIH databases has grown considerably. In addition to the first database, other compilations were published by MIT. These are intended for different areas of application. An important database, at least in the context of this thesis, is the MIT-BIH Normal Sinus Rhythm Database. This database includes 18 long-term ECG recordings of 5 men and 13 women, who were found to have had no significant arrhythmias. As in the original MITDB, all positions of the beats are annotated in this dataset. Since the shape of the R peak remains unchanged in both normal sinus rhythm and atrial fibrillation, this dataset is particularly well suited to benchmark peak detection on the BMBF dataset.

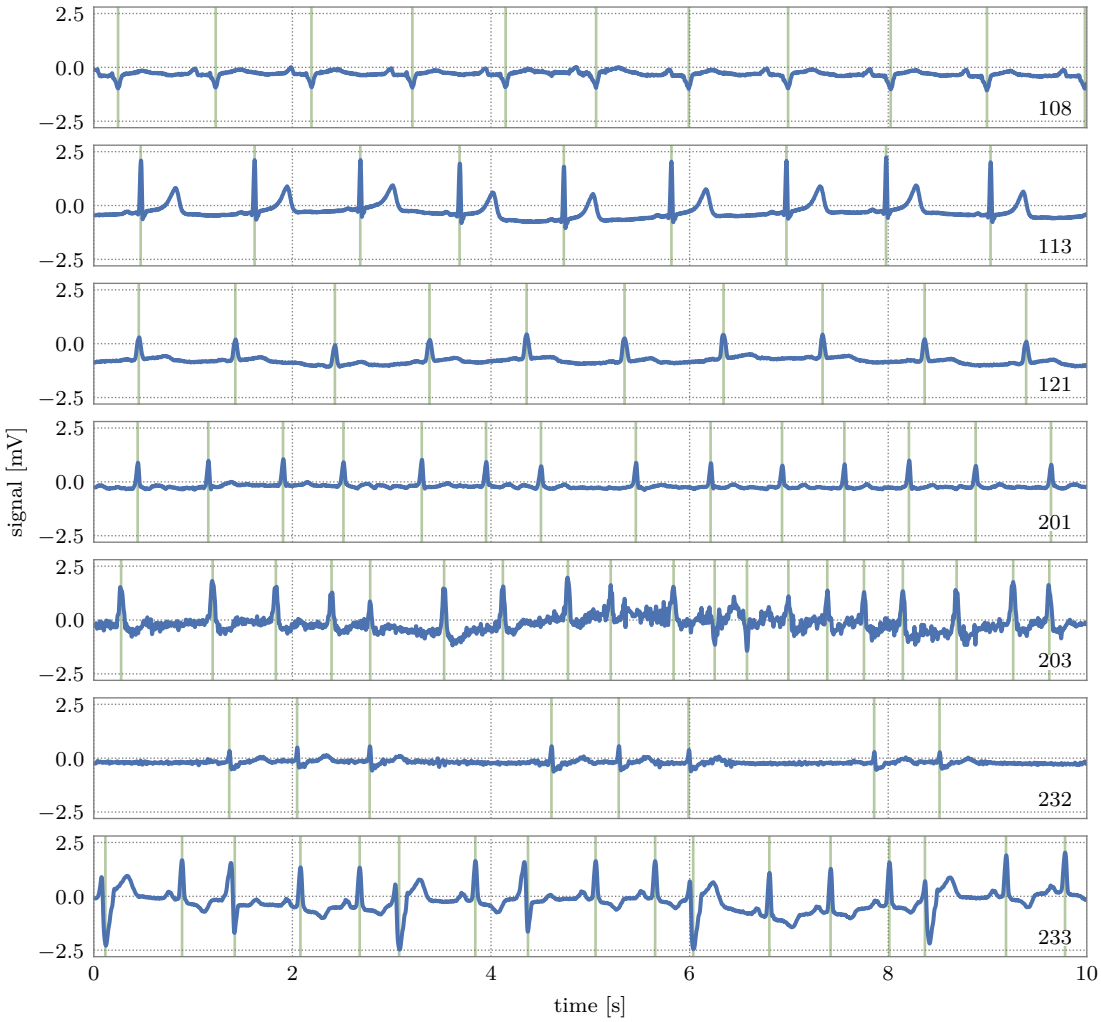

Figure 2.12: Some examples of ECG signals from MITDB with annotated beat positions. The first three traces (108, 113, 121) show normal sinus rhythm. Although they are all normal, they differ significantly in shape and amplitude. Trace 201 is an example of atrial fibrillation, the most common arrhythmia, the last three (203, 232, 233) show examples of other arrhythmias. Especially trace 203 is subject to considerable noise. The beat annotations were carried out by several experienced cardiologists and assign each beat to one of 19 different types.

2.3.2 Human Activity Recognition

Human activity recognition is also a suitable task for embedded low-power machine learning concepts. It can be used to monitor one's own physical activity throughout the day, but also in the elder care and health care sector such possibilities of monitoring are important as a supporting technology. Accelerometers and gyroscopes are a suitable data source here, since every modern smartphone has sufficiently precise sensors that can be utilized for this task.

2.3.2.1 The dataset

The used dataset of Anguita et al. (2013) contains data of 30 subjects, who perform activities of daily living while carrying a waist-mounted smartphone. Each subject performed a sequence of activities twice. The first recording was conducted with the device on the left side of their belt, during the second recording the subjects placed it on the belt to their liking. First they walked straight ahead, then up and down stairs. Afterwards they stood still for a while, sat down and lay down at the end.

Each sample in the dataset consists of linear acceleration and angular velocity signals in three dimensions, each. They were recorded on a Samsung Galaxy S II smartphone at a sample rate of 50 Hz. The public available database is already denoised and splitted into overlapping windows of 2.56 s length, i. e., they contain 128 data points. Each of these windows is labeled with one of the six different activities: walking, walking upstairs, walking downstairs, standing, sitting, laying. In addition, conventional feature engineering was applied to the windows resulting in 561 time and frequency features that were published for each window. However, these features are not used in this thesis. To allow a comparison of different approaches, the dataset was split into training and test sets, containing data from 21 and 9 subjects, respectively. Figure 2.13 shows examples from the test set for each of the six activities. Walking can be clearly distinguished from resting activities by means of the strong periodic movements, even by non-experts. But especially the differentiation between walking straight ahead and upstairs or standing and sitting is very difficult.

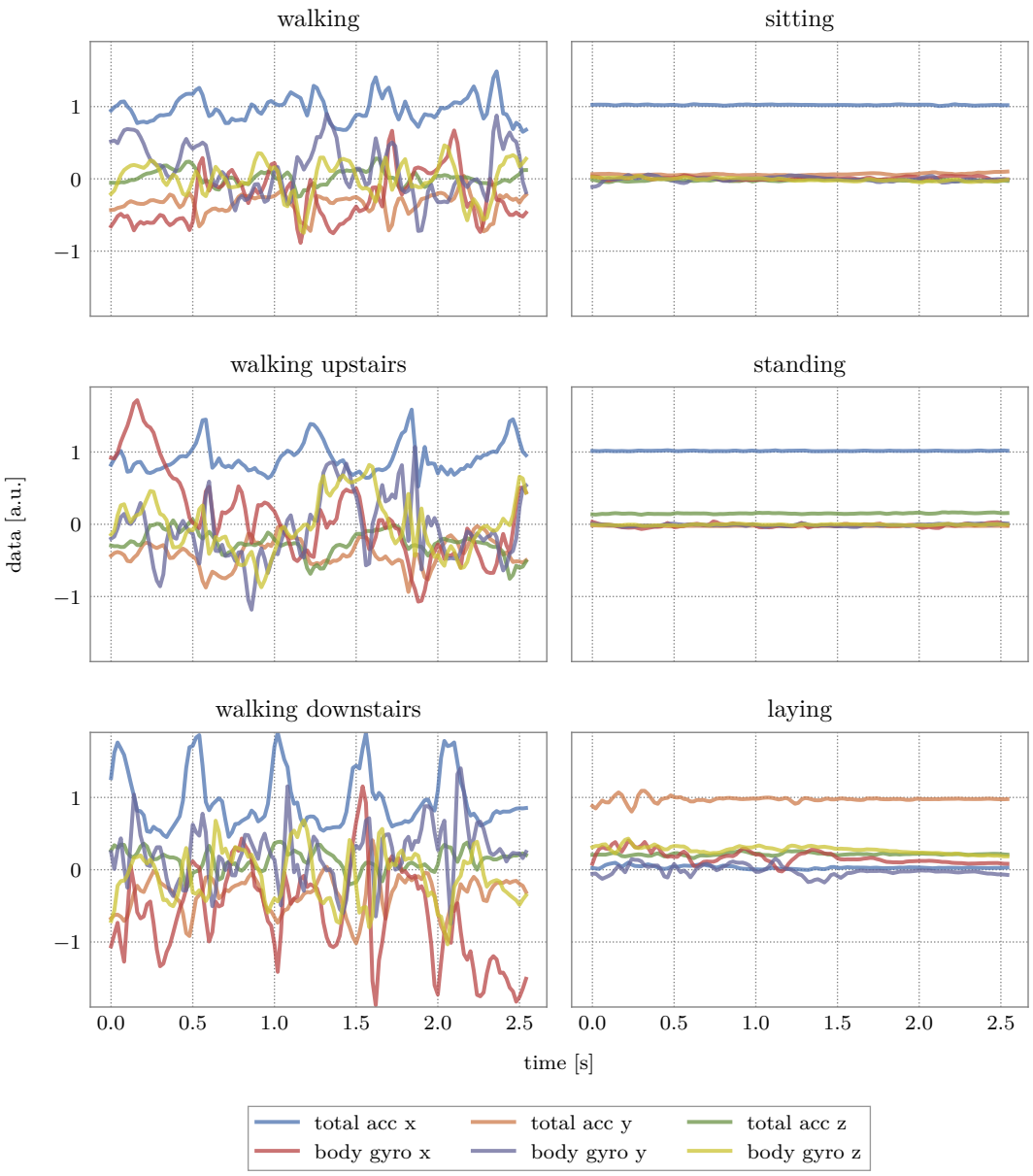

Figure 2.13: Some labeled examples from the *Human Activity Recognition* database. This database contains recordings from the accelerometer and gyroscope of a smartphone while performing activities of daily living. The data is already preprocessed and split into small chunks of 2.56 s length. Data taken from Anguita et al. (2013).

3 Methods and Tools

In this chapter the used methods and tools are presented. At first, different preprocessing steps are motivated. The selection includes typical steps in the processing of ECG recordings, but can also be applied to similar sensor data, such as the Human Activity Recognition example. These include procedures frequently used in the literature as well as own adaptations. This is followed by the artificial neural network models developed and used in the context of this thesis, which are able to classify based on the regularity of ECG traces.

3.1 ECG Preprocessing

Starting from the raw ECG signal, the next section discusses common preprocessing possibilities. The presented methods are typically executed one after the other, which leads to a chain of more and more abstracted data. Each element in the chain can form the starting point of an artificial neural network model for classification. The more processed the data is, the simpler the required networks tend to become. However, less preprocessing might be advantageous, since it is associated with additional computational effort. Furthermore, models on raw data are less task-specific and their architecture can be transferred more easily to other data. It is therefore important to find an optimum that requires as little conventional preprocessing as possible without making the network topology too complex.

An important aspect is that the multiplication mode of BrainScaleS-2 only has a digital input resolution of 5 bit. However, ECG recordings often cover a relatively large dynamic range, for the competition dataset presented in section 2.3.1.5 the inputs range from $-6\,\mathrm{mV}$ to $6\,\mathrm{mV}$, but the maximum amplitude of an R peak is only about $1\,\mathrm{mV}$. It should therefore be necessary in any case to significantly limit the dynamic range of the data. Figure 3.1 shows a sketch of a typical sequence of processing steps and the associated model complexity.

Many of the methods presented below operate on a floating local window on the data $y(x)$ at a position x with width w, which is defined as

$$\mathcal{W}(x; w) = \left\{ y(x - \tfrac{w}{2}), y(x - \tfrac{w}{2} + 1), \ldots, y(x + \tfrac{w}{2}) \right\} . \tag{3.1}$$

Using such a window is convenient for on-demand computing using embedded devices, as only a small part of the signal has to be kept in memory while processing. This reduces memory usage and latency in live processing a lot and might reduce computational complexity as well.

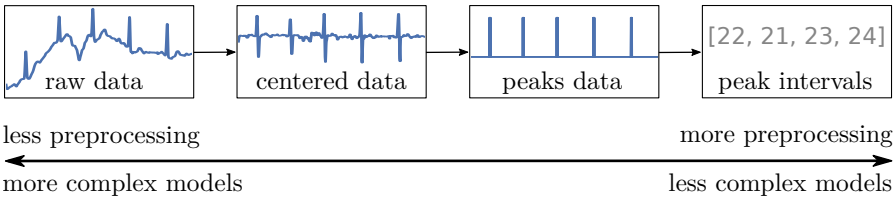

Figure 3.1: Overview of possible processing steps on ECG data and the associated model complexity. The generalizability of models operating on raw data is at the expense of their higher complexity.

3.1.1 Baseline Correction

A very common preprocessing on ECG data, as well as on other sensor data, is the subtraction of the baseline. Low-frequency changes in the signal are often caused by unrelated external influences and should therefore be removed. The goal is to preserve the specific form of the signal, in this case of the ECG features. For this purpose, it is important to filter out only the low-frequency baseline fluctuations without changing the shape of the high-frequency components. There are various options for this, all of the proposed ones have in common the determination of the baseline, which is then subtracted from the signal.

Floating Average The simplest approach to remove the baseline is to use a floating average filter on the signal. This can be written as

$$y_{\text{corrected}}(x) = y(x) - \frac{1}{w} \sum_{i=-\frac{w}{2}}^{\frac{w}{2}} y(x+i) \; . \tag{3.2}$$

The window size w is always a trade off between removing most of the baseline and preservation of the features of the signal. This works well on data that slowly varies compared to the typical width of its important features. A floating average is, e. g., used in Pandit et al. (2017).

Median Filter Another window based approach are median filters. Such a filter walks along the signal entry by entry, replacing each of them by the median of the surrounding window with size w. The baseline correction based on a median filter can be written as

$$y_{\text{corrected}}(x) = y(x) - \text{median}\{\mathcal{W}(x; w)\} \; . \tag{3.3}$$

Such a filter is not linear and therefore computationally more complex, since the values must be sorted within the window. However, the advantage of the median filter is that it cleanly removes outliers while adding no phase distortion. In contrast to a floating average the results are much more robust against variable baselines.

Morphological Filters Another method, proposed by Z. Liu, Wang, and B. Liu (2011), is the use of morphological filters. The morphological filters can remove peaks and dips of a specific width from the signal by combining successive opening and closing operations. Opening suppresses peaks, whereas closing is used to suppress dips. Combined open-closing and close-opening filter are defined as

$$\mathrm{OC}(y;k) = O(C(f;k);k) \quad \text{and} \tag{3.4}$$

$$\mathrm{CO}(y;k) = C(O(f;k);k)\ . \tag{3.5}$$

Their average is combined to a filter block, that removes all peaks and dips with a width of less than k from the raw signal y

$$\mathrm{OC\_CO}(y;k) = \frac{\mathrm{OC}(y;k) + \mathrm{CO}(y;k)}{2}\ . \tag{3.6}$$

The baseline removal can than be achieved with

$$y_{\text{corrected}} = y - OC\_CO(OC\_CO(y;k_1);k_2)\ . \tag{3.7}$$

For the filter widths the authors proposed $k_1 = 0.11\,\mathrm{s}$ and $k_2 = 0.27\,\mathrm{s}$, as these values correspond to the typical widths of QRS complex and T wave, respectively. This approach provides the most convincing results, but is also the most complex of the presented methods.

3.1.2 The Difference Method

This option is fundamentally different from the other baseline corrections in section 3.1.1 as it does not preserve the typical shape of the ECG. Instead, it uses a strided version of the discrete derivative of the signal, i. e., the value of a prior one is subtracted from each data point with a stride s

$$y_{\text{diff}}(x) = y(x) - y(x-s)\ . \tag{3.8}$$

This is also known as delta-encoding. Using the typical width of the R peak for the stride s, this method emphasizes the QRS complex and acts as a simple finite impulse response (FIR) filter. In terms of computational effort, this is even cheaper than the floating average, as only a few preceding values need to be stored. In addition, the use of this kind of delta-encoding in combination with clipping of negative values is insensitive to reversed electrodes in the recording.

3.1.3 Sample Rate Conversion

The data used in machine learning problems is typically sampled discrete in time or space. In this section the reciprocal distance between two adjacent discrete points of a signal is referred to as sample rate. This is to be regarded as both spatial and temporal, the presented methods can be applied to images and ECG recordings in the same way. The goal is to obtain a new discrete representation of the underlying continuous signal with modified sample rate.

Training samples like ECG recordings are often available at a quite high sample rate, for the datasets presented in section 2.3.1 it ranges from 256 Hz to 512 Hz. Since the human heart only has pulse rates in the order of 1 Hz, it might be advantageous to reduce the sample rate before the data will be processed by a neural network. This is especially useful when only differentiating between normal rhythm and arrhythmias, since the regularity of the heartbeat may already be sufficient for classification.

Upsampling, on the other hand, can be useful to unify data from different origins without losing too much of their characteristics or to classify data with machine learning models originally designed for higher resolutions.

Mean-Pooling Probably the simplest method, which takes all values of the original signal into account, is mean-pooling. In this method, the arithmetic average is formed over a window of width w, which reduces the sample rate by w^{-1}

$$y_{\text{meanpool}}(x) = \frac{1}{w} \sum_{i=-\frac{w}{2}}^{\frac{w}{2}} y(x+i) \; . \tag{3.9}$$

However, this method is not very suitable for ECG recordings, since these have narrow peaks with comparatively long pauses in between. The signal becomes blurred and it is difficult to distinguish the individual components.

Max-Pooling Another possibility is max-pooling. Here a value at the output corresponds to the maximum over a window with width w from the input. Normally w values are combined to a new value, so the sample rate is reduced by a factor of w^{-1}. Additionally, a larger window can be used in combination with a constant stride s to increase the width of the peaks in an ECG recording without losing additional resolution:

$$y_{\text{maxpool}}(x) = \max\{\mathcal{W}(x \cdot s; w)\} \tag{3.10}$$

The sample rate is hereby reduced by a factor of s^{-1}. This method preserves the amplitude of prominent parts of the signal, which results in a sharpened output. An advantage of an additional overlap is that the variance of otherwise very sparse inputs, such as ECG recordings, can be further increased, making the resulting signal at the neurons of the BrainScaleS-2 chip large enough to achieve sufficient resolution.

Maximum-Minimum-Difference An extension to max-pooling is the max-min-difference-pooling. Likewise the maximum, the minimum of the window is determined to derive their difference via

$$y_{\text{mmd}}(x) = \max\{\mathcal{W}(x \cdot s; w)\} - \min\{\mathcal{W}(x \cdot s; w)\} \; . \tag{3.11}$$

The decisive advantage is that negative contributions are also considered in this case. Applied to the raw ECG data, it is possible to process reverse polarity and baseline wander recordings. Even more robust results can be obtained by first applying the difference method from section 3.1.2, cf. figure 3.2.

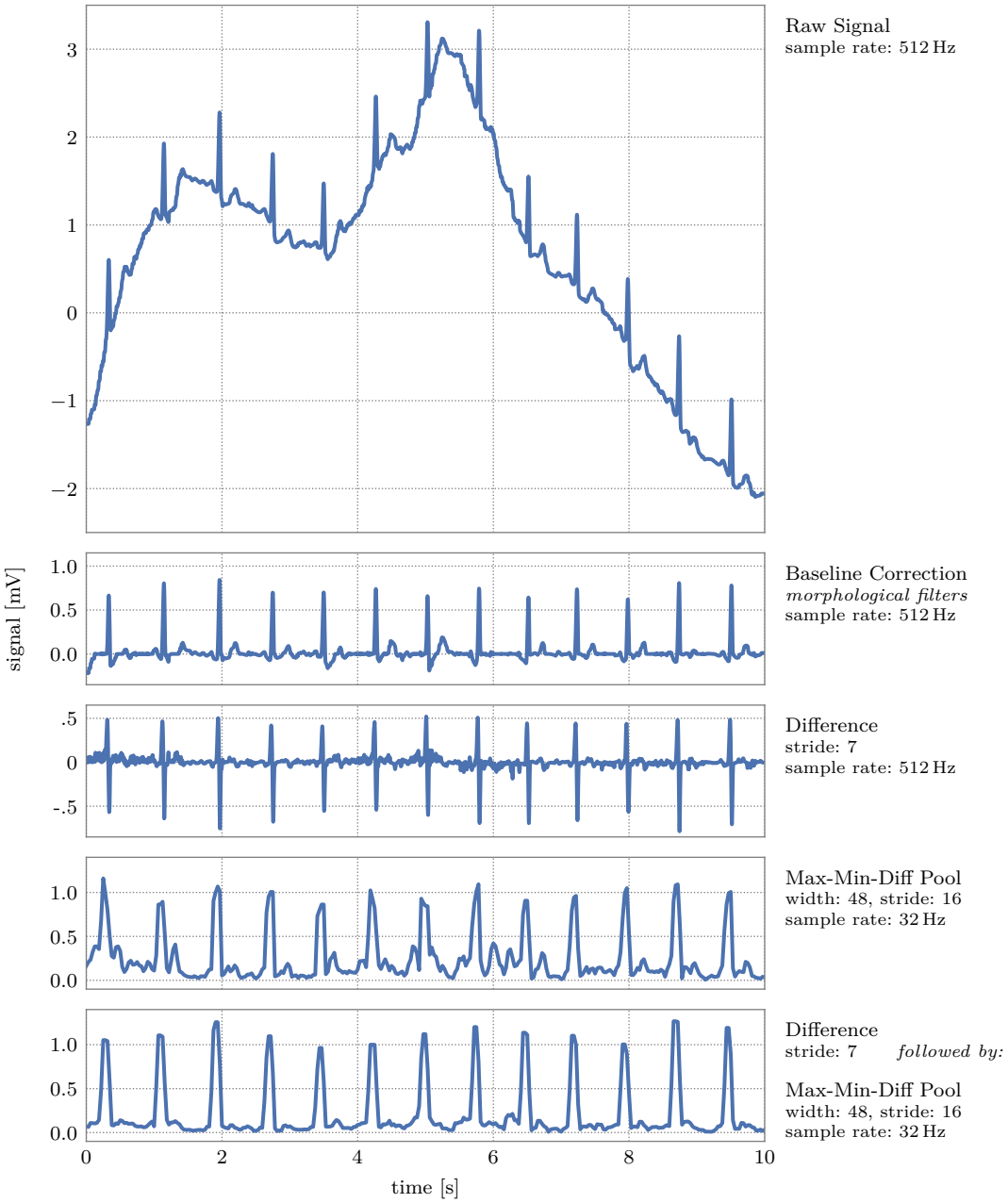

Figure 3.2: A subset of different preprocessing options. The raw signal (top) is affected by a fluctuating baseline. Using morphological filters this can be corrected in a very accurate way, preserving the shape. Alternatively, the presented difference method also provides a centered signal, requiring significantly less computing effort. The difference between min- and maxpooling is also applicable to the raw signal and reduces the sample rate considerably. However, the most convincing result is a combination of the difference method and min-max-difference pooling (bottom).

Trigonometric Interpolation This resampling method is very well suited for periodic signals and is a good choice if the input waveform needs to be preserved as much as possible. The signal is extrapolated using a polynomial of trigonometric functions, i. e., a linear combination of sines and cosines with appropriate periods and coefficients. This is chosen in a way that it goes through every data point of the signal.

In the case of equally spaced data points, fitting the parameters of such a polynomial corresponds to a discrete Fourier transformation. Hereby the data points x_j of the signal of length N are transformed to the Fourier coefficients $\hat{x}_k$, where

$$\hat{x}_k = \sum_{j=1}^{N} x_j \cdot \mathrm{e}^{-2\pi \mathrm{i} \frac{j-1}{N} k} \qquad \text{for} \quad k = 1, \ldots, N \,. \tag{3.12}$$

To now increase the length of a signal to $N' > N$, the coefficients are padded with additional zeros, i. e., $\hat{x}_k = 0$ for $k = N+1, \ldots, N'$. The data points of the interpolated signal are then obtained by means of the inverse transformation

$$x'_j = \sum_{k=1}^{N'} \hat{x}_k \cdot \mathrm{e}^{2\pi \mathrm{i} \frac{k-1}{N'} j} \qquad \text{for} \quad j = 1, \ldots, N' \,. \tag{3.13}$$

3.1.4 Beat Detection

As the presence of an arrhythmia can be inferred from the regularity of the heartbeat, conventional classification methods are often based on the positions of the beats. Among other possible applications, like detection of the heart rate, this is one of the reasons why the detection of heartbeats has been widely studied in literature. However, the robust detection of the beat positions from an ECG is a demanding task, as the amplitude of an ECG recording can vary strongly, especially between different measurements, but also within the same signal, cf. figure 2.12. In addition, different shapes of the *QRS* complex in other arrhythmias than atrial fibrillation make detection more difficult. In conjunction with baseline wander this makes it hardly possible to identify the beats with a fixed threshold or by their shape in the raw signal. Common methods in literature include Wavelet transformations (Madeiro et al., 2012; Übeyli, 2008), geometric feature extraction (Zhou, Hou, and Zuo, 2009) and principal component analysis (Rodríguez et al., 2015). Especially on mobile or embedded devices the use of complex filters or time-frequency domain transformations (Fourier or wavelet) is not easily possible due to limited computing power and memory availability. Elgendi et al. (2014) therefore presented a comparison of the complexity of different methods. In general there are different approaches, which are presented in the following.

Fixed Threshold A naive approach to determine the positions of the beats is to use a fixed threshold. If a certain threshold value is exceeded, a heartbeat is detected. But a certain amplitude may never be reached in one trace, which is already in the order of magnitude of the background noise in another trace. However, in combination with other techniques, such as delta-encoding (section 3.1.2) and max-min-difference (section 3.1.3),

this approach may achieve sufficient performance to enable classification. Since the heart rate is limited to about $300\,\mathrm{min}^{-1}$, it further improves accuracy to require a minimum distance of 200 ms between detected beats. This way especially prominent T waves are not recognized as additional beats, as their distance to the preceding QRS complex is only about 100 ms (Rangayyan, 2002).

Adaptive Threshold The problem of different amplitudes can be significantly reduced with an adaptive threshold, e. g., as suggested in Pandit et al. (2017). After a dead time, the threshold is adjusted depending on the amplitude of the preceding peak, after which it slowly decays (linearly) in time. If a higher value occurs afterwards, further conditions are checked within a window, if they match, the position is marked as peak.

However, datasets that contain a lot of noise and artifacts can be problematic. If, for example, a muscle contraction causes a sharp increase in the amplitude of an ECG recording, adaptive algorithms will adjust to this amplitude and are afterwards not susceptible to subsequent beats which may be several magnitudes smaller in amplitude.

Pan-Tompkins Algorithm A very popular method in ECG analysis was presented by Pan and Tompkins (1985). This extended adaptive threshold algorithm achieves outstanding results on the MIT-BIH dataset (section 2.3.1.7). 99.3 % of the detected beat positions match the manually annotated positions provided along with this dataset. It was initially developed for real-time analysis of ECG signals, but can also be applied as an offline version to already recorded traces. At first a bandpass filter is applied to avoid false detections caused by different types of inferences in ECG data. The algorithm than takes the first derivative, squares the signal and applies a moving average. The QRS complexes are then reliably detected using multiple thresholds defining the slope, amplitude and the peak width. They are dynamically adjusted during runtime and describe the signal better and better with ongoing processing.

This peak finder was originally implemented in assembly language, uses only integer arithmetic and was running on a Zilog Z80 microprocessor. So although this method is the most complex presented, the computing power of a small microcontroller is sufficient to analyze the ECG signals in real time.

Due to its high complexity, this algorithm is more robust than simpler methods based on adaptive thresholding. But, like these, it fails in the presence of strong disturbances that cause comparatively high amplitudes, as they occur in the competition dataset presented in section 2.3.1.5.

3.2 Classification Methods

The following section introduces some classification methods on ECG recordings, which can be used to perform this task with artificial neural networks on the BrainScaleS-2 hardware. Starting with conventional methods that require a comparatively large amount of preprocessing and may therefore not be ideal in terms of energy consumption or adaptability to other tasks and datasets. However, they are comparatively widely used and their function is easy to understand. Subsequently, the preprocessing is stepwise reduced and further methods developed in this thesis are presented. This way they become more complex and less task specific. The last model is quite general and can also be adapted to sensor data of a smartphone, differentiating between activities of daily living.

3.2.1 Plot of Successive RR Intervals

The inter-beat interval, i. e., the interval between two successive R peaks, is referenced as RR interval (cf. figure 2.9). The idea of plotting subsequent RR intervals against each other was proposed by Anan et al. (1990). A similar method to investigate complex periodicity in random signals was already used by Lorenz (1963). Therefore, this method is often referenced to as Lorenz-plot. Figure 3.3 shows an example of such a plot for a trace with regular beats and a trace with irregular rhythm. This classification method takes advantage of the fact that during atrial fibrillation the electrical conduction system in the atria is disturbed and the heartbeat is no longer triggered by the regular pulse of the sinus node. The distance between two successive pulses is therefore not constant, but almost randomly distributed within certain limits.

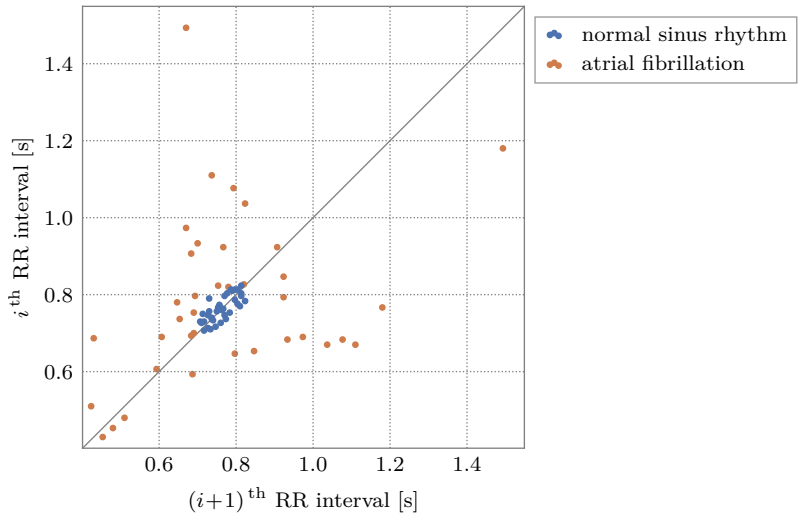

Figure 3.3: Plot of successive RR-intervals of two ECG samples, also known as Lorenz or Poincaré plot. Each interval is plotted over the previous interval. Regular intervals accumulate in a single region on the diagonal. In random sequences, as they occur in atrial fibrillation, the individual points are distributed over a large area, since there is no correlation to the previous interval.

3.2.2 CNN on Sparse Beat Positions

As shown in the previous section, the regularity of inter-beat intervals of the ECG recordings may already be sufficient to allow to distinguish between normal sinus rhythm and atrial fibrillation. Weis (2020) already presented two methods that perform the detection of regularity in spiking mode on the BrainScaleS-2 chip. Each of the two approaches takes the ECG data as a spike train with a spike on each heartbeat. For this, the positions of the beats must first be determined with one of the methods from section 3.1.4.

The second of the methods proposed in Weis (2020) uses the short-term synaptic plasticity (STP) circuits of the BrainScaleS-2 chip. These are used in spiking neural networks to regulate the input to a neuron in dependence of the time differences to previous spikes by adjusting the amplitudes of the currents to the neuron membranes. By taking the difference of excitatory and inhibitory connections with individual parameters to a number of different neuron populations, detectors for specific beat intervals can be obtained. If a matching interval is found, the membrane potential of a neuron is kept below the spike threshold and it remains inactive. A neuron population that does not fire for a certain period of time indicates a constant distance between the beats and therefore a normal sinus rhythm.

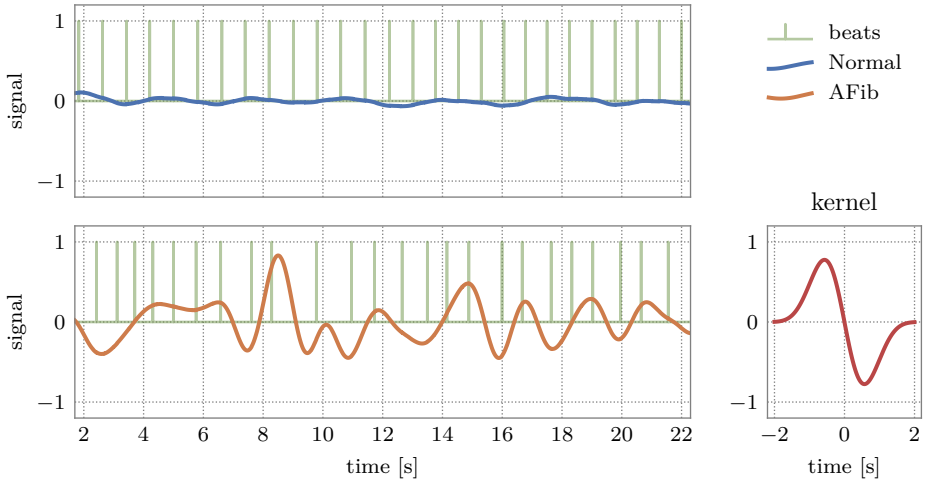

Figure 3.4: The preprocessing of sparse peaks with a convolution. First the positions of the beats must be detected. These are transformed into a vector, where the positions of the beats are 1, otherwise 0 (light green traces on the left). If these are then folded with a kernel (red; right), the signal disappears with regular beat intervals, whereas irregular data show strong fluctuations (left). This can now be further processed, for example with a small convolutional neural network.

The basic principle of the STP approach can also be handled in multiplication mode: a convolution with a sawtooth kernel, i. e., a vector with linearly incrementing values, corresponds to a measure for the distance to previous beat positions. Figure 3.4 shows

the result of the convolution with a slightly modified but equivalent kernel. This kernel is chosen such that the result is continuous and centered around zero, but this is not a requirement of this method. Again, a signal close to zero corresponds to a regular heart rhythm, strong fluctuations of the signal are an indicator for irregularities. This shows that with a reliable peak detection algorithm in combination with a convolutional neural network a distinction between regular and irregular beat intervals is possible.

3.2.3 CNN on Less Preprocessed ECG Recordings

Most of the published machine learning models to classify ECG recordings use techniques that are not yet and may never be supported by the BrainScaleS-2 software stack. For example some of the submissions to the Physionet Challenge 2017 (Clifford et al., 2017) use extreme gradient boosting (XGBoost), recurrent neural networks (RNNs) and long short-term memory (LSTM). However, the task set in the Physionet Challenge is more complex since a finer differentiation is required. Besides normal sinus rhythm and atrial fibrillation, the recordings had to be classified into two additional classes: other unknown arrhythmias and noise. But also convolutional neural networks are suitable for classification of ECG recordings, an impressive example was presented by Rajpurkar et al. (2017). The presented model consists of 34 convolutional layers and operates on ECG recordings with a length of only 1.28 s. It is able to differentiate between 10 different arrhythmias, normal sinus rhythm and noise, achieving a performance comparable to that of experienced cardiologists. These two examples do not set any requirements for low energy consumption; the last one is even more about a differentiated diagnosis. In portable, low power applications, however, it may be sufficient to detect the mere presence of an anomaly and consult a trained cardiologist in such cases.

Therefore a new deep neural network approach to classify ECG recordings is developed for the competition. The sole distinction between normal and arrhythmic heart beat simplifies the task considerably. Apart from meeting the minimum detection rate for arrhythmia patients of 90 % with a maximum of 20 % false positives, the energy consumption is rated and not the achieved accuracy. Thus it is possible and reasonable to reduce the complexity of the used convolutional neural network significantly.

The first major hurdle using raw data is the limited dynamic range of the input of BrainScaleS-2. The provided ECG recordings from the BMBF competition cover a dynamic range of $-6\,\mathrm{mV}$ to $6\,\mathrm{mV}$, but the maximum amplitude of the peaks is only about $1\,\mathrm{mV}$. With the input resolution of 5 bit the whole range can hardly be mapped without losing the comparatively small peaks. Because the baseline of the provided ECG recordings often varies significantly, it is hardly sufficient to process a small fixed range of the raw data values.

For these reasons a preprocessing has to be applied. When designing the following models for inference on BrainScaleS-2, it is assumed that the preprocessed data no longer show baseline fluctuations, are available at a reduced sample rate of 32 Hz and contain only positive values. Possible methods for preprocessing are presented in section 3.1 and figure 3.2. This makes it possible to use the input range and the link the BrainScaleS-2 hardware in an efficient way. The connection between memory and FPGA is significantly

faster than the subsequent connection from the FPGA to the BrainScaleS-2 chip. The downsampling factor of 16 was chosen such that both connections are almost saturated and the remaining sample rate of 32 Hz is still sufficient for inference.

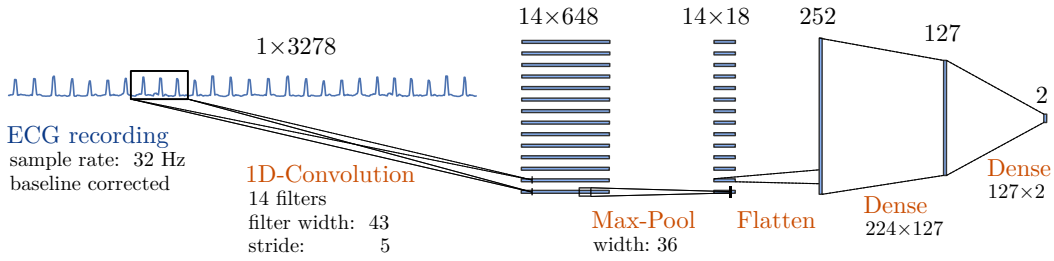

Figure 3.5: A possible configuration of a convolutional deep neural network for the classification of ECG recordings. The convolutional layer examines the input data using 14 different filters. The pooling layer is necessary to process longer recordings, a width of 36 is optimized for input lengths of 102.4 s and can be reduced accordingly to process shorter chunks of data. The classification is accomplished by two fully-connected layers with 127 hidden units at the end.

Layer	Activation	Input Shape	Output Shape	# of Params
Conv1d	ReLU	[1, 3278]	[14, 648]	616
MaxPool(36)	-	[14, 648]	[14, 18]	-
Linear-1	ReLU	[252]	[127]	32'131
Linear-2	Softmax	[127]	[2]	256

Table 3.1: Number of trainable model parameters and data shapes of the first proposed convolutional deep neural network on ECG recordings. The advantage of this proposal is that the weights can be written to the synapse array simultaneously and no reconfiguration is necessary. Each layer uses an additive bias. With a small weight matrix of 43×14 only a few parameters are needed in the first layer.

The model topology of a possible convolutional neural network model operating on minimal preprocessed ECG data on BrainScaleS-2 is shown in figure 3.5. It uses a one dimensional convolution layer with kernel size 43 and a stride of 5 for feature detection, followed by two dense layers. It requires an input length of 3278, which corresponds to 102.4 s for preprocessed data at a sample rate of 32 Hz. After the first layer, a max-pooling with a width of 36 is necessary in order to reduce the amount of input data for the second layer. This operation has to be realized in the digital part on BrainScaleS-2 with the SIMD microcontroller, but that may be rather costly as its complexity scales linearly. The hyper-parameters of all layers are optimized for the dimensions of the analog substrate to achieve a balance between accuracy, energy efficiency and execution speed. This makes it possible that all the weights fit at the same time on the two synapse arrays of 128×256

values each. There is no need to reconfigure the weight matrix during inference, which saves valuable time and thus optimizes energy consumption.

The number of trainable parameters and the data shapes are listed in table 3.1. In addition to the multiplicative weights, additive biases are used in each layer. These might allow to compensate for possible offsets on the analog hardware.

3.2.4 Human Activity Recognition

The solution to detect abnormalities in ECG recordings with a convolutional neural network is quite general and can be applied to completely different data without substantial adjustments. Therefore, the ability of inference with a one-dimensional convolutional neural network on the BrainScaleS-2 chip can be additionally demonstrated with the human activity data set introduced in section 2.3.2.

The model used and some early results on the first hardware version of the BrainScaleS-2 chip have been already presented in Spilger, Müller, Emmel, et al. (2020). The only differences to the ECG model presented in section 3.2.3 and sketched in figure 3.5 are the larger number of input channels of 6 (acceleration and angular velocity in three dimensions each) and 6 output neurons, since with walking, walking upstairs, walking downstairs, staying, sitting, and laying six different classes must be distinguished. Since the input vectors of this dataset are much shorter, it is further possible to omit pooling between the layers.

Layer	Activation	Input Shape	Output Shape	# of Params
Conv1d	ReLU	[6, 128]	[16, 16]	3'072
Linear-1	ReLU	[256]	[125]	32'125
Linear-2	Softmax	[125]	[6]	756

Table 3.2: Number of trainable model parameters and data shapes for the Human Activity Recognition example. The convolutional layer has more parameters than the corresponding layer of the ECG model, since it features a separate kernel for each of the 16 input channels.

The provided dataset from Anguita et al. (2013) is already denoised and split into windows of 2.56 s in length, so hardly any additional preprocessing is necessary. However, since BrainScaleS-2 cannot process negative inputs in the default configuration, the values have to be moved into the positive range and scaled to the maximum input value range of the matrix multiplication mode (cf. table 2.1).

4 Software Implementation

4.1 The Processing Framework

During this work, a data processing framework was developed. Although its main purpose is to be used for preprocessing ECG recordings in classification experiments, it has a very general structure, which allows experiments with other time-based datasets as well.

4.1.1 Design Principles

This section discusses concepts that were considered during the development of the framework and why they are important. Those universally applicable concepts of modern software development can make a crucial difference in research by ensuring the quality and maintainability of experiment code.

Object-oriented programming During development, special attention was dedicated to the logical structure of the code. Since the introduction of the first object-oriented programming language in 1972 (Kay, 1993), the paradigm has inspired many programming languages such as C++ and Python. Nowadays, object orientation is widely used in modern software development. It structures the code by grouping related properties and behaviors into individual objects. A decisive advantage is consequent abstraction: an object offers an interface of operations, but it is not predetermined how exactly the associated functionality is implemented. Inheritance of classes makes it possible to take over the entire functionality of the parent and to extend it with additional methods and attributes. This results in a natural relationship between the objects, basic functionalities become more and more specialized and extended. In addition to the logical context, this also automatically ensures a high degree of reuse of existing code. The experiment code makes extended use of object orientation and is therefore easy to understand and adapt.

Test-driven development The whole experiment code is excessively unit-tested. More than 99 % of the lines of code are executed during testing. The test cases not only cover common usage examples but also tests for reasonable errors in the case when something does not fit together. The tests were all created before or during the writing of the actual code. This helps to define the required functionality and interface before any code is written and ensures that it is fulfilled by the actual implementation afterwards. A disadvantage, however, is that writing the tests is time-consuming. It happens that the code of a test is even more extensive than the later implementation of the functionality. An experiment by George and L. Williams (2004) showed that test-driven development took 16 % more time but produced overall higher quality code.

In case of restructuring the code, the presence of tests has a very positive effect. Since the tests ideally cover the entire range of use cases, ideally there is no need for manual testing to determine whether the results are identical. In addition, unit-testing allows very detailed tests. When a test fails, this makes it easier to identify the origin of the error. Through continuous integration, it becomes possible to automatically execute all tests of the code including all dependent projects as soon as changes occur. This allows to effectively build and maintain a reliable code base, even for large projects with many dependencies.

Annotated types In most programming languages variables and functions are typed. This defines, e. g., the value range of an object and ensures that only allowed operations can be performed, which prevents runtime errors. A differentiation can be made between statically typed programming languages, whose type checks already take place at compile time, and dynamically typed languages, whose types are specified during runtime with the first assignment. Python is dynamically typed, but type indications can be set according to `PEP 484` (Rossum, Lehtosalo, and Langa, 2014). The main advantage of annotated types is the possibility of static code analysis. It ensures that all components really do fit together at crucial points. In addition, the type information significantly increases the readability of the code for a human reader and facilitates refactoring. Type hints can be seen as a very simple and elegant way to document the method signature. But despite type hints Python remains a dynamically typed language, which means a high degree of flexibility.

Static code analysis and code style To detect errors in the source code even before execution, a static code analysis can be performed. Therefore every line is statically analyzed by a so called linter, Pylint (Python Code Quality Authority, 2020b) in this case. It is checked for the existence of used functions and variables in the current context and the correct implementation of interfaces. Thus, these errors are detected directly when writing the code and can be easily corrected. In addition, the code is examined with Pycodestyle (Python Code Quality Authority, 2020a) for compliance with the style recommendations according to `PEP 8` (Rossum, Warsaw, and Coghlan, 2001). This especially includes the line length, code block indentation, and meaningful names. Thereby the code remains clear and consistent, avoiding code smell. However, refactoring an existing code base to make it conform to the style constraints of the code checker can be a tedious task. Furthermore, since Python is a dynamically typed programming language with multi inheritance, there are complex cases that are not covered cause linting to fail.

Code review 100 % of the experiment code is subject to a review process. At least one additional person goes through the code changes of every commit. A major advantage is that code review often results in cleaner, better documented code, because a second person who may not be involved in the development process verifies the comprehensibility of the change. The programmer has to critically scrutinize every part of his work and legitimate changes with reasonable functionality. Code review encourages the exchange

of information and skills between developers. Besides personal education this improves future code quality. In addition, it increases the number of people who are familiar with the codebase and if individual developers leave the project, it can still be continued.

A disadvantage is the associated time delay. It often takes several weeks before a commit is reviewed, and possibly additional weeks until it can finally be merged to the main code base. Code review is time-consuming because the reviewer needs to understand the function and motivation of the code and often needs a thorough in-depth knowledge of the context. Despite of this, it has been shown that code review coverage shares a significant link with software quality (McIntosh et al., 2014), so it is an important tool to develop reliable and maintainable code.

Documentation Each internal and external function of the API has a written documentation that explains its task and all of its parameters. The same applies to the implemented classes and interfaces. All documentation is compatible to Sphinx (Brandl, 2020), a widely used Python documentation generator. Originally developed for the documentation of Python itself, Sphinx makes it easy to create comprehensive documentation, not only for python projects. By using the reStructuredText markup language it is easy to link to other parts of the documentation and websites, give code examples and highlight important features.

4.1.2 Components

The developed processing framework is structured by means of object orientation into individual components, the most relevant concepts are briefly introduced in the following.

4.1.2.1 Samples

All the elements in a time-based experiment dataset have a lot of properties in common, therefore all sample classes inherit from a common abstract base class called `TimeSample` (cf. figure 4.1). It has methods to replace individual properties, create empty samples and convert the samples to a tuple of data and label, making them convenient to use in machine learning frameworks. It also provides abstract methods to allow concatenation, slicing and plotting of the samples. Like images and videos, time-based samples often contain multiple channels. These can generally be scalar signals from different sources, e. g., different leads in an electrocardiogram. But also the components of vectorial sensor outputs can be treated in this way.

A `TimeseriesSample` contains equally spaced data at a certain sample rate. Such samples are used, for example, in the ECG and Human Activity Recognition datasets (section 2.3) to hold raw or processed continuous data. In contrast to this, a `SpikingSample` contains the binary information if a certain event occurs on a discrete point in time. This is useful for the beat positions in ECG datasets, but also for spiking data as used in inference with spiking neural networks.

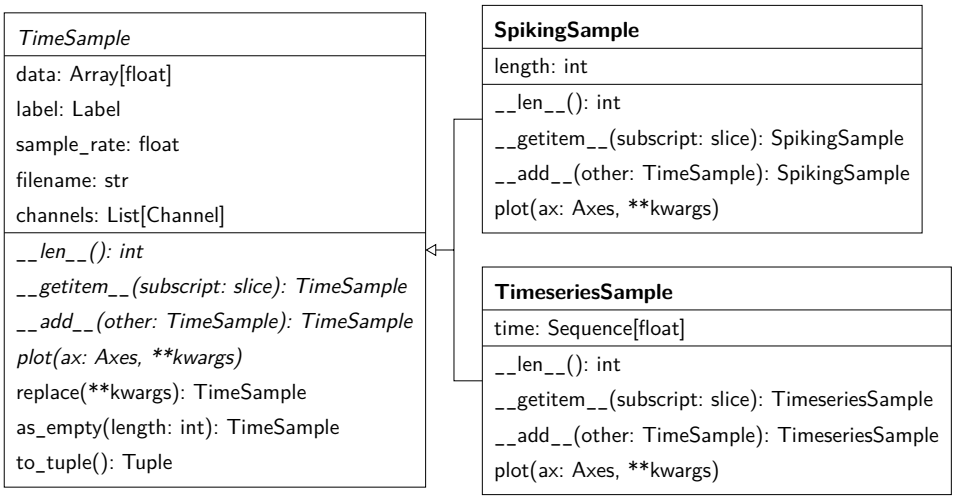

Figure 4.1: Class diagram of the implemented sample classes. `SpikingSample` and `TimeseriesSample` inherit from a common abstract base class `TimeSample` which defines most of their properties and some general methods. Classes, properties and methods in italics are abstract.

4.1.2.2 Datasets

A `Dataset` is a container which can hold an arbitrary number of samples, which may generally be of different lengths and types. Since the dataset class implements the `Sequence` interface, instances can be used without further modifications with all common machine learning frameworks including PyTorch.

4.1.2.3 Dataset Loaders

Most of the datasets come in different data formats. Typical ones are text files with comma separated values, serialized MATLAB arrays or pure binary data, possibly with an additional text header, or even audio formats. In addition, the data may come in very different data types, e.g. `int8`, `uint16` or `float32`. The task of a `DatasetLoader` is to load and streamline the raw data so that it can be easily processed afterwards. The result is a dataset with continuous or spiking samples. Many datasets are provided with separate training and test sets in order to achieve comparability between publications. If available, these can also be accessed separately by a data loader to prevent them being mixed up.

Loaders for different sources of ECG recordings, i.e. from the *BMBF Competition* (BMBF, 2019), the *Physionet 2017 Challenge* and the *MIT datasets* (Moody and Mark, 2001) are implemented, as well as loaders for other time-based data, i.e. the *Human Activity Recognition* dataset (Anguita et al., 2013) and the *Heidelberg Digits* (Cramer et al., 2019).

4.1.2.4 Transformations

`Transform` is a transformation that can be applied to datasets. There is a wide range of transformations that fulfill very different tasks. The fact that the transformation is applied to the entire dataset and not to individual samples has the advantage that the number or ordering can also be changed. For example, it is possible to filter the samples or split them into smaller chunks. Especially splitting can be very useful for time data, as it allows you to train on shorter samples from different points in time instead of a single long one.

The `Compose` transformation allows to combine arbitrary transformations by executing them one after the other. This way it allows individual preprocessing steps to be easily grouped together. Furthermore, there is indexed access and by means of Python slicing it is possible to apply only a subset of the transformations.

4.1.3 Caching

All dataset loaders and transformations implement the `UniquelyIdentifiable` interface. As a consequence, each instance of these classes has a universally unique identifier (UUID). This is based on the output of the SHA-1 hash algorithm applied on the source code of the class, the source code of all base classes and the state variables of the instance. Changes that may produce different results thus reliably generate a different UUID. If this is used as a key, it is possible to cache the result of the complete preprocessing chain.

A concatenation operation was developed to link the UUIDs of the individual parts of the chain. The remarkable feature of this operation is that it is associative, but in general not commutative, except for a neutral element id_e. This means

$$(id_1 \circ id_2) \circ id_3 = id_1 \circ (id_2 \circ id_3) \qquad \forall\ id_1, id_2, id_3 \tag{4.1}$$

$$id_e \circ id_1 = id_1 = id_1 \circ id_e \quad \text{and} \tag{4.2}$$

$$id_1 \circ id_2 \neq id_2 \circ id_1 \qquad \forall\ id_1, id_2 \neq id_e\ . \tag{4.3}$$

This is implemented as a matrix multiplication of 4×4 matrices. The 128 bit value of each UUID is mapped as 8 bit unsigned integers to the entries of a matrix, the two resulting matrices are matrix-multiplied and than converted back to an UUID. However, this implementation does not exclude inverse UUIDs, so there is a theoretical possibility that equation (4.3) does not hold. Nevertheless the probability to hit an inverse is assumed to be in the order of magnitude of the probability of a hash collision and is therefore neglected.

This kind of implementation makes it possible to cache parts of the chain and reprocess only when changes occur. This makes, e. g., parameter sweeps much easier and more efficient, since potentially long processing times can be eliminated. The whole caching algorithm is encapsulated by a wrapper for arbitrary dataset loaders, which is constructed with the desired transformation and a path to a caching folder. This makes the use of caching for transformed datasets particularly easy and convenient.

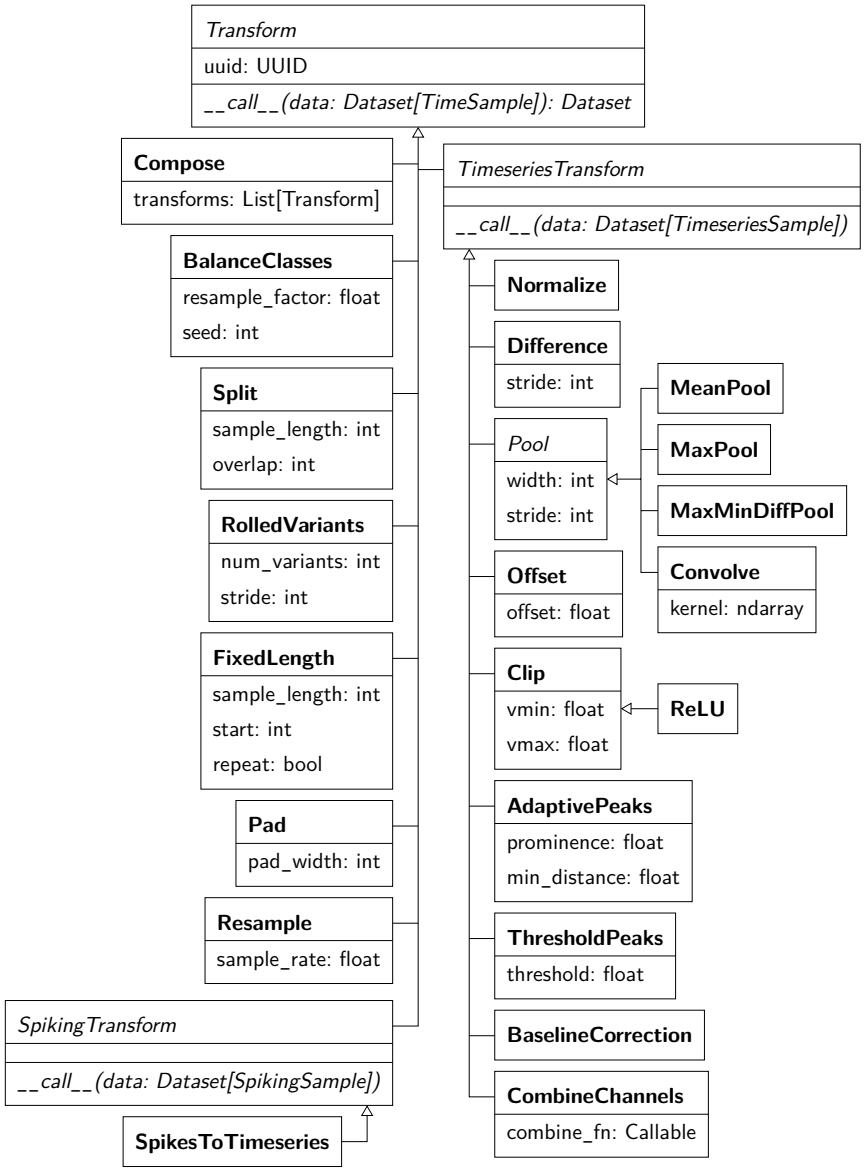

Figure 4.2: Class diagram of most of the implemented transformation classes. The classes in the top left can operate on arbitrary datasets. The transformations on the right will only work on timeseries samples, as they require the continuous nature of the samples to transform.

4.2 `hxtorch` – PyTorch on BrainScaleS-2

In the context of this thesis some features and improvements were contributed to `hxtorch` presented in Spilger, Müller, Emmel, et al. (2020). The software is open-source and most of the features in this section have already been released on github[1]. The idea of `hxtorch` is to provide a high-level interface for the multiplication mode, and to utilize the existing functionality of PyTorch to use BrainScaleS-2 as an analog accelerator for inference with artificial neural networks. The strategy during the implementation was to complement the existing operations of PyTorch with further operations which are executed on BrainScaleS-2. No parts of PyTorch are replaced, so its complete functionality is preserved.

4.2.1 Backward Pass for Training with Hardware in the Loop

When training artificial neural networks, the weights are iteratively adjusted based on the gradient of a loss function, as already described in section 2.1.2. To provide the gradient for every weight by backward propagation, each of the applied operations in the network needs an implementation of its backward pass. This function defines the gradient of the operation and calculates results for all input parameters based on the arguments of the operation and the gradient of the subsequent layer.

For the multiply accumulate operation on BrainScaleS-2, the forward pass is approximated with a simplified linear relationship:

$$y_i = \sum_j x_j \cdot w_{ij} \cdot g_{\text{BSS-2}} + \kappa_i \quad \text{with} \quad \kappa_i \sim N(0, \sigma) \; . \tag{4.4}$$

The analog noise κ follows a normal distribution N with standard deviation σ. The amplification factor $g_{\text{BSS-2}}$ is assumed to be constant for each neuron, which may not necessarily be the case. This approximation results in the gradients

$$\frac{\partial y_i}{\partial w_{ij}} = x_j \cdot g_{\text{BSS-2}} \quad \text{and} \quad \frac{\partial y_i}{\partial x_j} = w_{ij} \cdot g_{\text{BSS-2}} \; . \tag{4.5}$$

Unlike the inputs x_j and the weights w_{ij}, the factor $g_{\text{BSS-2}}$ is not known in general. Due to the analog nature of the system and the way the accumulation of the signals on the neuron membranes is implemented, the value of $g_{\text{BSS-2}}$ slightly differs not only between different neurons but also for different inputs.

Neglecting the gain in equation (4.5), i. e., assuming $g_{\text{BSS-2}} = 1$, would preserve the sign of the gradient and just scale it. This could therefore be compensated for in single-layer networks by adjusting the learning rate accordingly. With several successive layers, however, any deviation increases exponentially. Since the backward signal will be inappropriately scaled by a factor $g_{\text{BSS-2}}$ in each layer, the propagated signal will be scaled by $g^l_{\text{BSS-2}}$ after l layers. Because of $g_{\text{BSS-2}} \ll 1$, this leads to diminishing signals, whereby the first layers are not adjusted and the algorithm does not converge. Especially in deep neural networks

[1] `https://github.com/electronicvisions/hxtorch`

with a large number of successive layers, this becomes a serious problem. This can be solved by determining the value of the gain factor for each multiplication individually. For this purpose, when execution a multiply accumulate operation on the chip during training, it is simulated simultaneously on the host computer. The factor $g_{\text{BSS-2}}$ can now be determined from the ratio of the output of the operation on the hardware $\boldsymbol{y}_{\text{BSS-2}}$ and an equivalent matrix multiplication

$$g_{\text{BSS-2}} = \frac{1}{J} \sum_j \frac{y_{\text{BSS-2},j}}{\sum_i x_i \cdot w_{ij}} \,. \tag{4.6}$$

4.2.2 The Convolution Operations

The BrainScaleS-2 chip only supports plain vector-matrix multiplications. But since discrete convolutions can be transformed into single vector-matrix operations, convolution operations become possible as well. Basically there are two different approaches to implement this efficiently, both of which have their advantages and disadvantages.

For the first method, the weight matrix is kept stationary and is successively multiplied by an increasingly shifted part of the input. For this purpose a strided view of the input vector is created, from which each element is multiplied by the kernel (left of figure 4.3) This was implemented in collaboration with Philipp Spilger, who particularly designed the reshaping in the beginning. The advantage of this method is that the kernel is always placed at the same position in the synapse array. Deviations, which still occur after calibration due to fixed-pattern noise, can be effectively compensated with adjusted weights through in-the-loop training. Since the size of the kernel in typical convolutional neural networks is usually significantly smaller than the maximum dimensions that fit on the synapse array (cf. table 2.1), a significant amount of space remains unused, resulting in longer runtime and reduced energy efficiency.

Another possibility is to arrange the weights multiple times in the synapse array. For this, strided versions of the kernel are placed next to each other in a larger matrix, as sketched at the right in figure 4.3. This way the input has to be multiplied only once with the resulting matrix, combining the beforehand single operations into a big one that is executed in parallel. This reduces the execution time during inference up to the factor of the number of parallel executions. The downside of this approach is that it is no longer possible to individually compensate fixed-pattern noise of the substrate. Since the weights are placed simultaneously in different synapses, modifications due to training with hardware in the loop are not able to adapt to the hardware differences of individual synapse drivers, synapses and neurons.

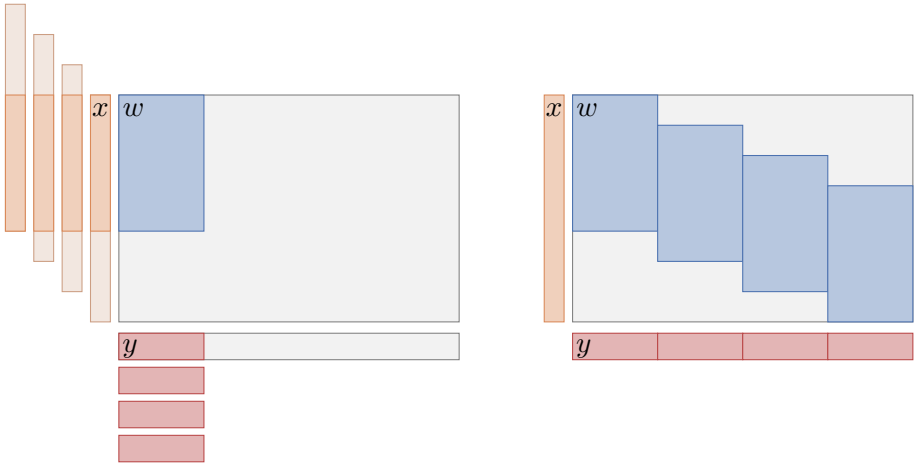

Figure 4.3: Two different variants of the one-dimensional convolution operation on BrainScaleS-2 implemented in `hxtorch`. On the right: the weight w is placed at a fixed position on the synapse array, the input vector x is multiplied by the kernel in several shifted variants to perform the convolution. Since the kernel is always in the same place, fixed-pattern deviations of the analog substrate could be compensated by adjusting the weight. Left: in order to maximize the usage of the synapse array, the same weight w is rolled out several times on the synapse array. This reduces the number of necessary MAC operations on the hardware by the factor of how often the kernel fits side by side into the physical dimensions of the chip. The runtime for the convolution operation is reduced by this factor, especially for small kernels with few channels this makes a significant difference. However, due to deviations of the individual components of the analog hardware, slightly different results of the combined small operations occur, since the same weights are applied in parallel on different parts of the hardware.

4.2.3 The Mock-Mode – Simulating the Hardware

The execution of operations on BrainScaleS-2 from a host computer is several orders of magnitude slower than using vanilla PyTorch operations (cf. figure 4.4), especially on the HICANN-X v1 chip due to a bug described in more detail in Weis et al. (2020). Furthermore, only a limited number of BrainScaleS-2 hardware setups are available. In order to be able to develop quantized models compatible with BrainScaleS-2 using conventional CPUs or GPUs in reasonable time, an operation mode was developed which simulates some of the behavior of the hardware.

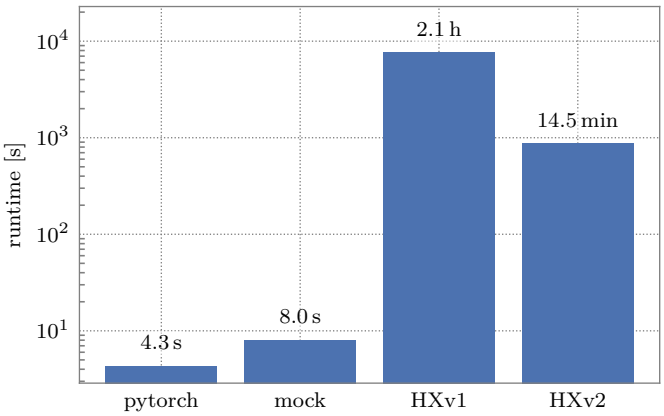

Figure 4.4: Comparison of the runtime for training a small CNN model as presented in section 3.2.4 for one epoch. Due to quantizing of all values in every step, reshaping and adding additional noise, the mock-mode has almost twice the runtime than the original PyTorch implementation in this case. However, the training on hardware takes even more time, the runtime increases in the order of multiple magnitudes. Especially the first chip version stands out because of a bug in the hardware design which was fixed in the v2 revision. All measurements were conducted using the same host computer. It should also be noted that the relative runtime varies significantly depending on the batch size, the complexity of the model, the number of CPU cores used, the network load and much more. This plot is therefore not an in-depth analysis but rather an impression of a typical runtime during training.

The so-called mock mode uses the same pre-processing of inputs and post-processing of outputs as the operations performed on BrainScaleS-2. The only difference is that the actual vector-matrix multiplication from equation (2.21) is simulated by PyTorch operations. In particular, this means that all values are rounded to integer numbers and clipped to the possible value ranges of BrainScaleS-2, see table 2.1. As only matrices with a maximum height of 128 can be processed natively on the hardware, multiplications with larger weight matrices are divided into smaller parts by an underlying backend, individually calculated on the hardware and digitally added at the end. To reproduce this behavior in mock mode, all matrices are first divided into slices with a maximum height

of 128 entries. On these segments the actual operation is then performed individually and the results are added up again afterwards. As in section 4.2.1, the vector matrix multiplication of the input x_j and the weight w_{ij} is simulated using

$$y_i = \sum_j x_j \cdot w_{ij} \cdot g_{\text{BSS-2}} + \kappa_i \quad \text{with} \quad \kappa_i \sim N(0, \sigma) \; . \tag{4.7}$$

For the noise κ a normal distribution is assumed. Both the gain factor $g_{\text{BSS-2}}$ and the standard deviation of the noise σ can be adjusted via global parameters during initialization.

In conventional machine learning frameworks, additional noise layers are used to reduce overfitting during training. However, unlike in this case, the noise of the hardware also reduces the resolution during inference, which is why the noise is not deactivated in this case either. Nevertheless, such a behavior can be easily achieved with deactivating the noise of the mock mode for evaluation by setting its standard deviation σ to zero.

4.2.3.1 Differences Between Mock Mode and BrainScaleS-2 Hardware

In order to keep things simple and to achieve reasonable runtime in mock mode, it uses a linear approximation of the multiply accumulate operation on hardware, cf. equation (4.7). Thereby only statistical noise is simulated. On the BrainScaleS-2 chip, however, there are significant differences between the individual neurons despite calibration. This fixed-pattern noise is shown in figure 4.6. A uniformly filled vector was multiplied 100 times by a uniform matrix of the same size as the synapse array. For each neuron the mean value and the standard deviation of the output is shown. A fixed-pattern deviation of about 5 % to 10 % can be observed, depending on the used chip, calibration and the input values. The statistical deviations are about 2 LSB, therefore this value is a reasonable default for the standard deviation of the noise in mock mode. Since the fixed-pattern deviations are not simulated in mock mode, in most cases further training with hardware in the loop is necessary in order to achieve sufficient accuracy on BrainScaleS-2.

Another important aspect is the linearity of the multiply accumulate operation. Figure 4.5 shows a comparison of the achieved neuron output as a function of weights and inputs for the execution on the analog chip and the corresponding simulation in `hxtorch` mock mode. A variation of the weights shows a quite linear relationship within the limits of the analog-to-digital converter. Minor deviations in linearity can be observed at negative weights. When varying the inputs for small values, the bottom part of figure 4.5 shows a significant deviation of the hardware results from the simulation. The magnitude of the output of the used chip is already higher than expected at low inputs. It will have to be investigated to what extent this influences the inference and whether these effects can be compensated by in-the-loop training.

It should be noted that uniformly filled matrices are used and therefore the results are averaged over all synapses. This may hide other individual deviations of the analog circuits.

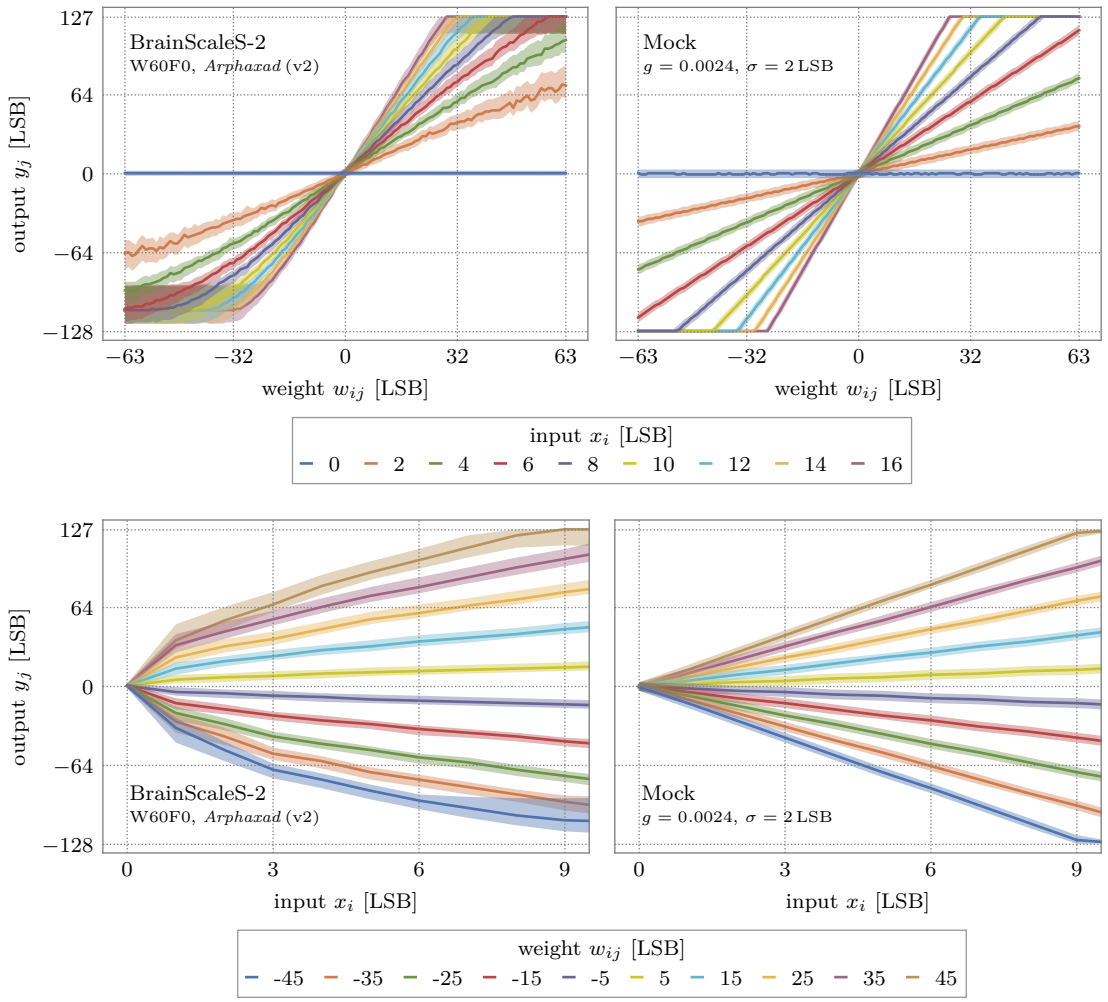

Figure 4.5: Comparison of the deviations in terms of linearity between mock mode and execution on BrainScaleS-2. A uniformly filled vector and a uniform matrix are multiplied 100 times for each combination of x_j and w_{ij}, the median outputs y_i of all neurons are shown, the colored regions comprise 95 % of the corresponding neuron outputs. Top: within the available output range, there is usually a good linear relationship between weight and output value. Slightly more pronounced deviations can be observed for negative weights. Bottom: especially for small inputs the relation between input value x_j and output y_i is not as linear as would have been expected from the linear approximation. This leads to significant differences to the results from the simulation in this value range (right).

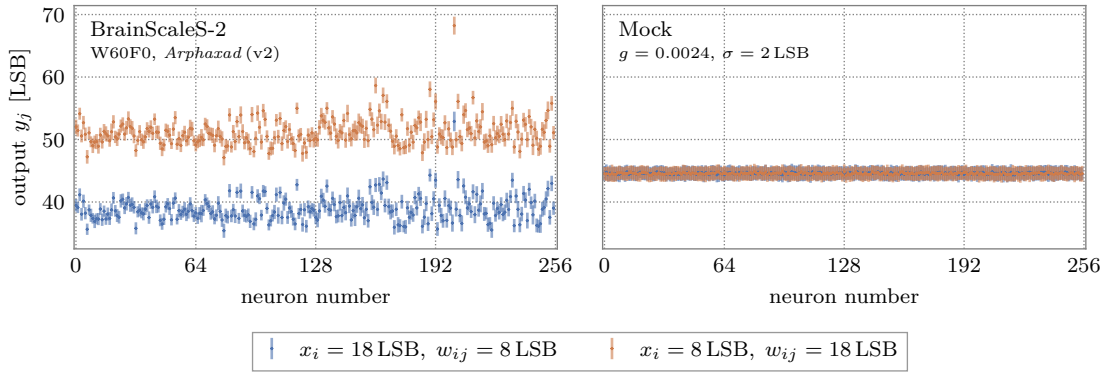

Figure 4.6: Comparison of statistical noise and fixed-pattern deviations between mock mode and execution on BrainScaleS-2. A uniformly filled vector and a uniform matrix are multiplied 100 times, the mean output of each neuron and its standard deviation are shown. Left: results of the hardware runs with an input of 18 LSB and weight of 8 LSB (blue). The mean standard deviation of the neurons is 1.89 LSB, the fixed-pattern deviations between neurons are about 5.6 %. Swapping the values results in significantly higher output (orange). Right: the same calculation simulated with the mock mode. No fixed patterns are included, the mean value and the statistical noise are in good agreement with the results on BrainScaleS-2. Swapping the input values has no effect.

4.2.4 Replacements for the PyTorch Layers

To use the operations provided by `hxtorch` as part of an artificial neural network, additional layers were implemented in the `hxtorch.nn` module. These complement the existing layers in the PyTorch module `torch.nn`, but use the operations of `hxtorch` and allow to set the additional hyper-parameters like `num_sends` and `wait_between_events`. An optional argument allows to activate the mock mode in each layer individually.

All `hxtorch` layer implementations use the same 32 bit float weights and biases as their respective counterpart from PyTorch. At each run, the weights may be scaled to the dynamic range of BrainScaleS-2, i. e., −63 LSB to 63 LSB and rounded to the closest integer value. This way of quantization ensures maximum compatibility with existing optimizers in PyTorch, as these still process the float representation of the weights.

Another possibility would have been to not implement additional standalone layers and to use the existing PyTorch layers instead. Monkey patching of the original operations can then allow to still use the `hxtorch` implementation with these layers. As a consequence, each layer would have to use these overwritten versions of the operation. In addition, any further matrix multiplication would implicitly be executed on BrainScaleS-2. But that may be unwanted, since some calls to these operations could be unrelated to the actual inference and, e. g., only used to calculate a confusion matrix. In addition, the hardware parameters and the mock switch would be global, since they can no longer be selected per-layer as argument. Introducing separate layers for the `hxtorch` operations on the other

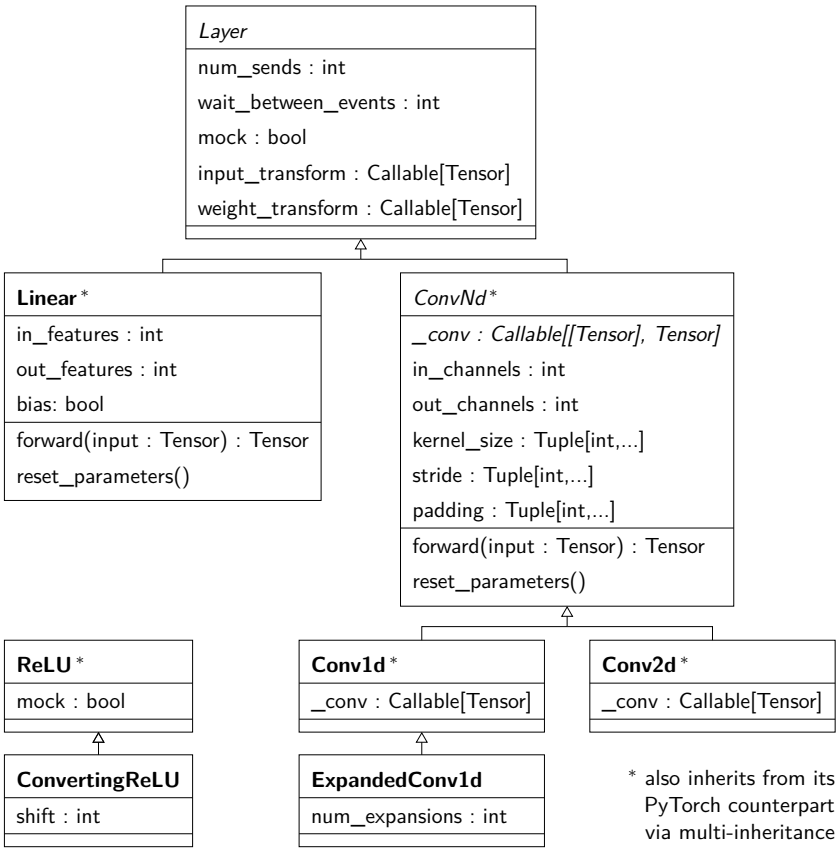

Figure 4.7: Class diagram of the implemented additional layer classes in `hxtorch`. These form the building blocks for the neural network models with PyTorch, cf. listing 4.1. The *Layer* introduces additional hardware parameters, using num_sends the gain can be scaled individually. Furthermore the inputs and weights of each layer can be transformed before every forward pass and thus be adapted, e. g., to the input range of BrainScaleS-2 during training. Linear, Conv1d, and Conv2d use `hxtorch` operations to replicate the functionality of their PyTorch counterparts. ExpandedConv1d is a replacement for the Conv1d and allows a parallel execution on BrainScaleS-2 by combining its weights. ConvertingReLU extend ReLU by shifting the 8 bit output activations to the allowed input range. Classes, properties and methods in italics are abstract.

hand makes it easy to execute some of the layers on BrainScaleS-2 while the rest runs on the CPU or GPU, cf. listing 4.1. This is especially useful when debugging a deep neural network and to use features currently unsupported by the BrainScaleS-2 implementation. Separate layer implementations also allow to adjust the hardware parameters and thus the gain g on the chip individually for each layer of the network.

Figure 4.7 shows the class diagram of all additionally implemented `hxtorch` layers. Each layer has a PyTorch counterpart, whose interface is implemented and extended. Additional, hardware parameters and input and weight transformations can be specified. Furthermore, ExpandedConv1d is an extension to Conv1d, which combines several vector-matrix multiplications and executes them in parallel, as described in section 4.2.2. As they inherit from their PyTorch equivalents and use the same weights and biases, all trainable parameters of native PyTorch layers can be loaded and handled. This makes it naturally possible to execute and train a model pre-trained in pure PyTorch on BrainScaleS-2.

```
import torch.nn as nn
import hxtorch.nn as hxnn

class Model(nn.Module):
    """ An example model using both torch and hxtorch layers """

    def __init__(self):
        self.features = nn.Sequential(
            hxnn.ExpandedConv1d(
                in_channels=3, out_channels=2,
                stride=5, kernel_size=15,
                num_expansions=15,
                num_sends=5, mock=False),  # hardware parameters
            hxnn.ConvertingReLU(),
        )
        self.classifier = nn.Sequential(
            hxnn.Linear(
                128, 32, num_sends=2,
                mock=True),  # enable simulation
            hxnn.ReLU(),
            nn.Linear(32, 2),  # floating-point precision
            nn.Softmax(),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.shape[0], -1)  # flatten
        return self.classifier(x)
```

Listing 4.1: Example of a model using `hxtorch` layers. All of them can be seamlessly combined with vanilla PyTorch components. For each layer the mock mode can be selected individually. Additional hardware parameters such as `num_sends` are passed to the layers as additional arguments. All other parameters are identical to those of the corresponding PyTorch counterpart.

4.2.5 Initialization

Since the multiplication mode of BrainScaleS-2 only supports fixed value ranges (cf. table 2.1), it is even more important that the amplitudes of the signal remain as constant as possible throughout the network. Due to the quantization, smaller values quickly lead to very low resolution, whereas too high values result in saturation of the neuron membrane. Therefore, all weights of the `hxtorch` layers are by default initialized using the Kaiming method as proposed by He et al. (2015) and introduced in section 2.1.4.2, which is also used by PyTorch by default.

Due to the nature of the multiply accumulate operation on the analog hardware, this method has to be adapted. On one hand, the multiplication in first approximation adds another additional factor, the gain $g_{\text{BSS-2}}$, which was already discussed in section 4.2.3. In addition, the output range is different from the input range, i. e., $-128\,\text{LSB}$ to $127\,\text{LSB}$ vs. $0\,\text{LSB}$ to $31\,\text{LSB}$, which is why the output is typically scaled by a factor of $g_{\text{scale}} = 4$ after the ReLU is applied. Altogether, the total gain to be considered is therefore

$$g_{\text{tot}} = \frac{g_{\text{scale}} \cdot g_{\varphi}}{g_{\text{BSS-2}}} \,. \tag{4.8}$$

Using this result, the weights are initialized according to a uniform distribution $\mathcal{U}(-r, r)$, whose boundaries are defined as

$$r^{(l)} = g_{\text{tot}} \cdot \sqrt{\frac{3}{n_{\text{in}}^{(l)}}} \,. \tag{4.9}$$

4.2.6 Move Pretrained Models to `hxtorch`

Since the individual layers inherit from the corresponding pytorch implementations, the weights can be transferred directly to the corresponding `hxtorch` layer, e. g., with

```
layer2 = hxtorch.nn.Conv2d(1, 2, 3)  # equivalent hxtorch layer
layer2.load_state_dict(layer1.state_dict())  # get weights from layer1
```

However, when transforming existing models that have been trained with floating point precision to `hxtorch`, there are some aspects to consider. In contrast to floating-point arithmetic, quantization with fixed steps and limited resolution allows only limited value ranges, cf. table 2.1. Furthermore, the current implementation provides a rather fixed factor $g_{\text{BSS-2}}$ between input and output which can only be varied to a limited extent between different layers.

This means that the weights have to be scaled to be within the allowed value range of BrainScaleS-2. To benefit from the full resolution available, it is also desirable to utilize the whole range from $-63\,\text{LSB}$ to $63\,\text{LSB}$. On the other hand it is important that the outputs of the operation match the correct range. To additionally boost the outputs the hardware parameter `num_sends` can be used. It allows the inputs to be sent to the neuron several times in a row before the results being read out.

5 Results

The following section contains a summary of the results for the classification of ECG recordings and human activity recognition. The processing framework presented in section 4.1 is used for loading, transforming, caching and plotting of all datasets. This, especially, includes the preprocessing steps from section 3.1, like baseline correction, pooling and beat detection. The machine learning models for the classification of these data are realized with the components from PyTorch. For quantized training in mock mode and execution on the BrainScaleS-2 hardware these are accompanied by the extensions provided by `hxtorch` (cf. section 4.2).

The actual execution of the experiments is handled with Sacred (Greff et al., 2017). Sacred is a versatile tool to configure, organize, log and reproduce computational experiments. In the implemented workflow it is used for the initialization of the hardware, i. e., automatic calibration, and the parameterized run of an experiment. In particular, the used software version, all run parameters, and the complete log output is collected and saved to disk. After each training epoch, important metrics, i. e., timestamps, loss, accuracy and confusion matrix, and the trained weights are stored for later evaluation. This makes it possible to re-run or continue a previous training session, maybe even with changed parameters or using a different backend.

5.1 ECG Classification

The approaches presented in this thesis can be divided into two categories: those that infer the class from the positions of the beats and those that operate on almost raw data. In terms of universality, the latter have clear advantages, as they can be easily adapted to very different tasks. This can be seen in the section 5.2, where it is shown that such a model can be applied to acceleration sensor data to infer activities of daily living.

5.1.1 Beat Position Based Models

In order to achieve reliable results with beat-position-based methods, an accurate detection of the beats is crucial. Since most of the MIT databases contain annotations of the beat positions and are widely used in literature since their first release in 1982, they provide an excellent benchmark for beat detection algorithms. Table 5.1 shows the performance of the used prominence-based peak finder. For this purpose, the difference method (see section 3.1.2 and figure 3.2) is combined with a prominence-based peak finding algorithm from SciPy (Virtanen et al., 2020). The MIT normal sinus rhythm database is used as a benchmark, since atrial fibrillation does not affect the QRS complex and this dataset contains only normal beat shapes. With a detection sensitivity of more than 99 % in most

recordings, the applied method is on par with algorithms used in literature, a comparison of other methods is given in Pandit et al. (2017).

Tape no.	TP beats	FP beats	FN beats	Sensitivity [%]
16265	27518	136	1	100.00
16272	18397	673	40	99.78
16273	25165	4	4	99.98
16420	25880	7	5	99.98
16483	27845	7	6	99.98
16539	23474	17	3	99.99
16773	22796	2	2	99.99
16786	24021	0	0	100.00
16795	25283	5	40	99.84
17052	21897	9	2	99.99
17453	26156	31	6	99.98
18177	27583	94	20	99.93
18184	25096	8	4	99.98
19088	27686	795	129	99.54
19090	23316	68	4	99.98
19093	20859	51	0	100.00
19140	26423	69	0	100.00
19830	28623	344	3443	89.26

Table 5.1: Performance of the used peak detection method, benchmarked on the MIT normal rhythm database (Moody and Mark, 2001). TP, FP, and FN denote the numbers of true positive detections, false positives and false negatives, respectively. The sensitivity is defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$.

Evaluation of the BMBF Dataset

The peak detection algorithm is applied to the training data of the BMBF competition presented in section 2.3.1.5. The left part of figure 5.1 shows the detected inter-peak intervals. From these a mean heart rate of $72.45\,\mathrm{min}^{-1}$ for healthy patients and $75.85\,\mathrm{min}^{-1}$ for patients with atrial fibrillation can be determined.

A histogram of the differences between successive RR intervals is shown on the right side in figure 5.1 for normal sinus rhythm and atrial fibrillation. As expected, the two classes differ significantly. While in normal sinus rhythm the subsequent intervals are quite similar and vary little, atrial fibrillation shows a broader distribution, which indicates the random nature of the heartbeat.

Even a small fully-connected neuronal network with 20 hidden units is sufficient to meet the accuracy requirements of the competition. From the histogram of peak interval deviations (Figure 1.4, right), 93.6 % of the recordings with atrial fibrillation and 89.7 % of the recordings with normal sinus rhythm can be labeled correctly using the BrainScaleS-2

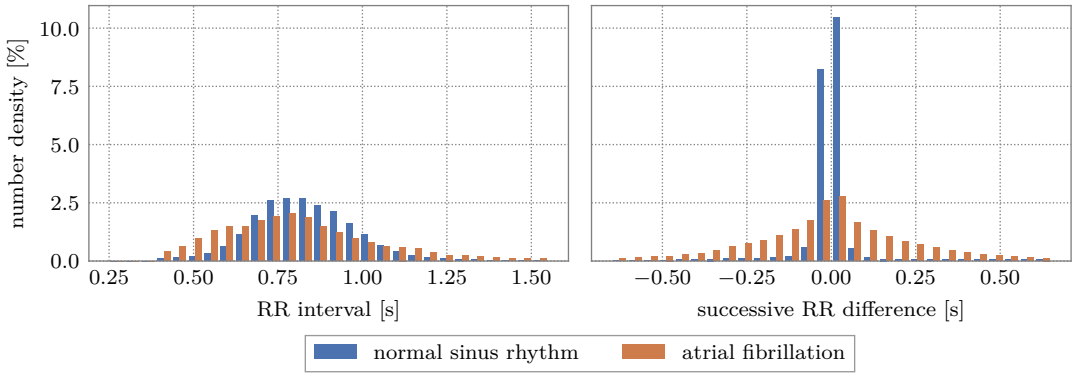

Figure 5.1: Histogram of the obtained beat-to-beat-intervals (RR intervals) and the difference of successive RR intervals of the BMBF competition training set, separated in normal sinus rhythm and atrial fibrillation. The peak of the distribution of R peak distances (left) is in the expected range of 0.6 s to 1.0 s for healthy subjects, corresponding to pulse rates of $60\,\mathrm{min}^{-1}$ to $100\,\mathrm{min}^{-1}$. For atrial fibrillation the distribution is broader and the beat intervals are shorter on average. As expected, the differences in the duration of successive intervals (right) show a comparatively broad distribution for atrial fibrillation. This random behavior is an indicator for an arrhythmia.

hardware. However, since the preprocessing used is quite complex and task-specific and not suitable for other non-ECG datasets, this method will not be investigated further.

The chaotic nature of atrial fibrillation can be observed in the Lorenz plot in figure 5.2. Successive intervals are hardly correlated, so that the points for atrial fibrillation are distributed over a large area. Normal sinus rhythm, on the other hand, is concentrated at the diagonal, a changing heart rate causes a gradual motion up or down this line.

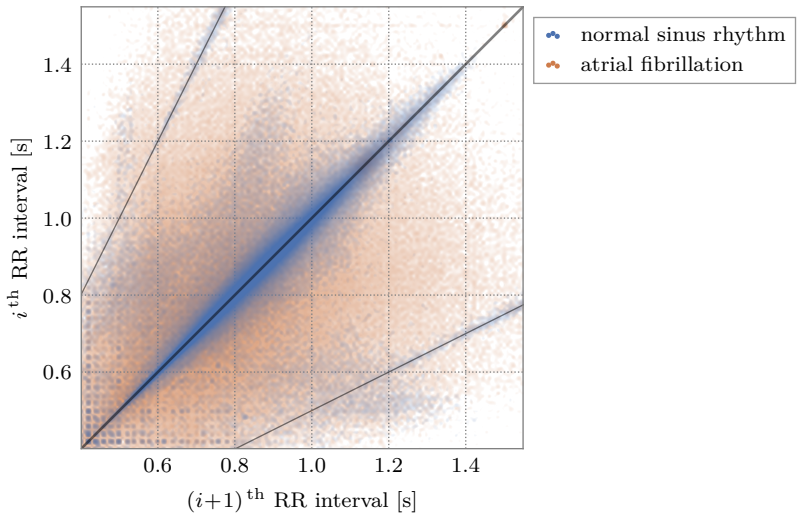

Figure 5.2: Lorenz plot from the detected beat positions in training set of the BMBF competition. Points on the diagonal correspond to two consecutive intervals of equal length. Atrial fibrillation is spread over the entire area of the plot, which implies chaotic behavior. Samples with normal sinus rhythm are more concentrated in the middle. An accumulation can also be seen on the two lines below and above, which correspond to twice the interval and indicates an undetected beat in between. The regular dotted pattern in the lower left is due to power-line inference at 50 Hz.

5.1.2 Less Preprocessing: Convolutional Neural Network Models

When training a convolutional neural network on ECG recordings, the choice of appropriate preprocessing has shown a decisive impact on a successful result, especially the choice of the dynamic input range that effectively reduces the influence of perturbations. For all results in this section, a difference-encoding with a stride of 7 (≡ 14 ms at 512 Hz sample rate) is applied to the raw ECG recordings, followed by a maximum-minimum pooling with a width of 48 (≡ 94 ms) and a stride of 16 (≡ 31 ms). The result of these steps has a sample rate of 32 Hz and is subsequently limited to values from 0.1 mV to 1.0 mV. The individual parameters are selected in such a way that the R peaks are amplified, the baseline is corrected and the temporal resolution is sufficient to allow a reliable classification. An ECG recording processed this way is shown in figure 5.3.

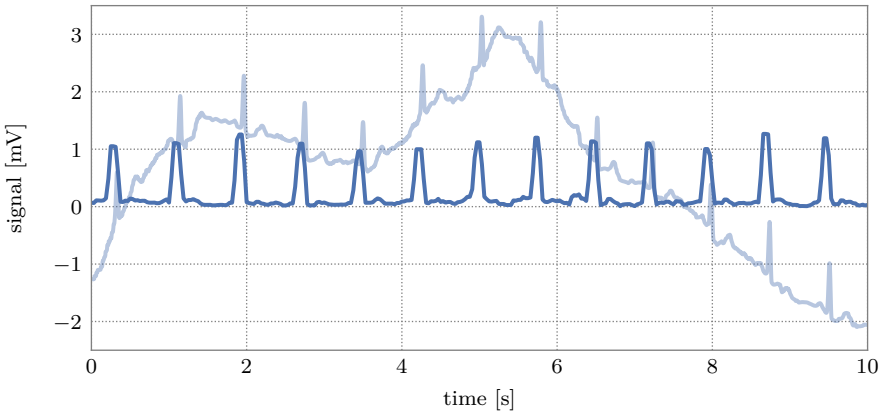

Figure 5.3: An example of a preprocessed ECG recording for use in the CNN based models. First, a discrete derivative with a stride of 14 ms is applied, followed by maximum-minimum-difference pooling to reduce the sample rate to 32 Hz. Original ECG recording taken from the *PhysioNet Challenge 2017* (Clifford et al., 2017).

5.1.2.1 The First CNN Model

An overview of the structure and the on-chip arrangement of the first convolutional neural network already presented in section 3.2.3 are shown in figure 5.4. The hyperparameters of the model are chosen such that all used weights can be written to the two synapse arrays of the chip at the same time, thus no reconfiguration is necessary during inference. The model uses a one-dimensional convolution with a stride of 5 and operates on input lengths of 102 s. To enable this input length, a pooling operation is used after the first layer, which combines 36 values to their respective maximum. This can be accomplished either with the integrated processor, i. e., the PPU, or the FPGA. It scales linearly in the PPU an cannot be further optimized, but using the FPGA on the other hand results in an increase of the latency to classify a single ECG recording.

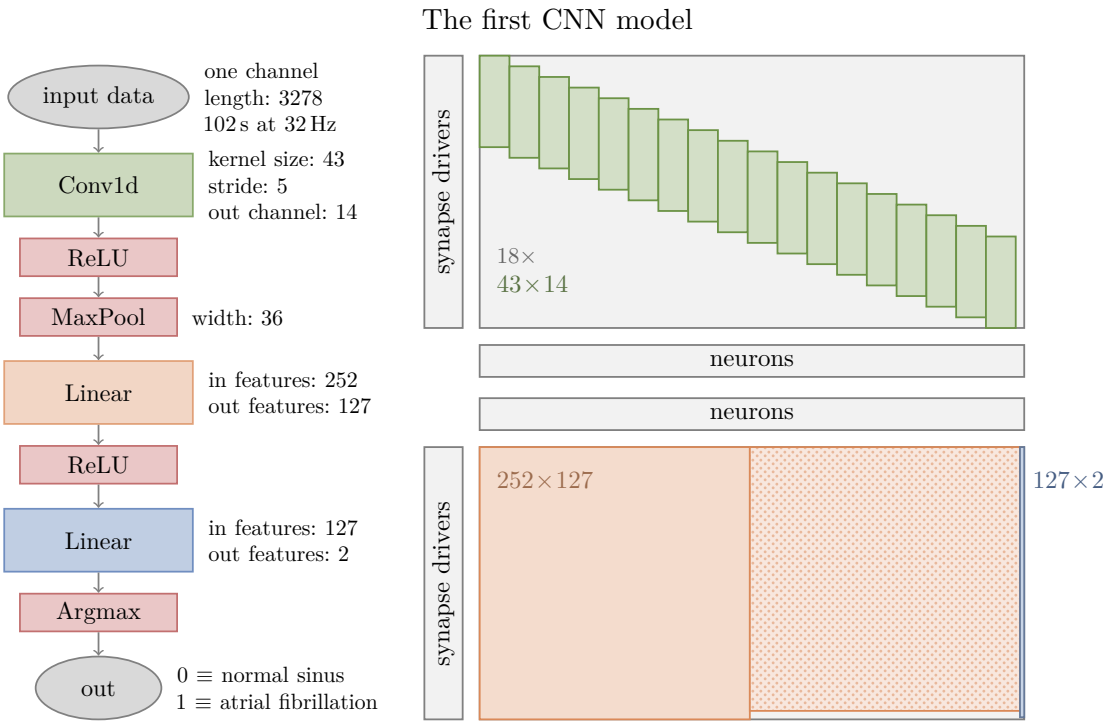

Figure 5.4: Layer structure (left) and on-chip arrangement (right) of the first convolutional neural network model already sketched in figure 3.5. The convolutional layer (green) is processed in the upper synapse array, the identical weight is arranged 18 times on the substrate to enable parallel processing. All rectified linear units and the maxpooling after the first layer (red) are performed in digital logic by the PPU. The further processing takes place on the lower synapse array with a fully connected layer and 127 hidden neurons (orange). It has to be split and arranged side by side, because it does not fit into the array in one piece. The dotted part of the layer uses a different label bit and therefore recieves the second half of inputs at the same time and is processed in parallel. The actual classification is then achieved in the last layer (blue) with two neurons on the right, which form the output.

Figure 5.5 shows the loss and the recall accuracies for both classes during training on the BMBF dataset. Thereby the standard PyTorch operations are used, which operate with floating-point precision. The quantized training in simulation and with BrainScaleS-2 are performed with the equivalents from `hxtorch`. With the method for initializing the weights presented in section 4.2.5 it is possible to achieve the required accuracy both in simulation and using the BrainScaleS-2 chip.

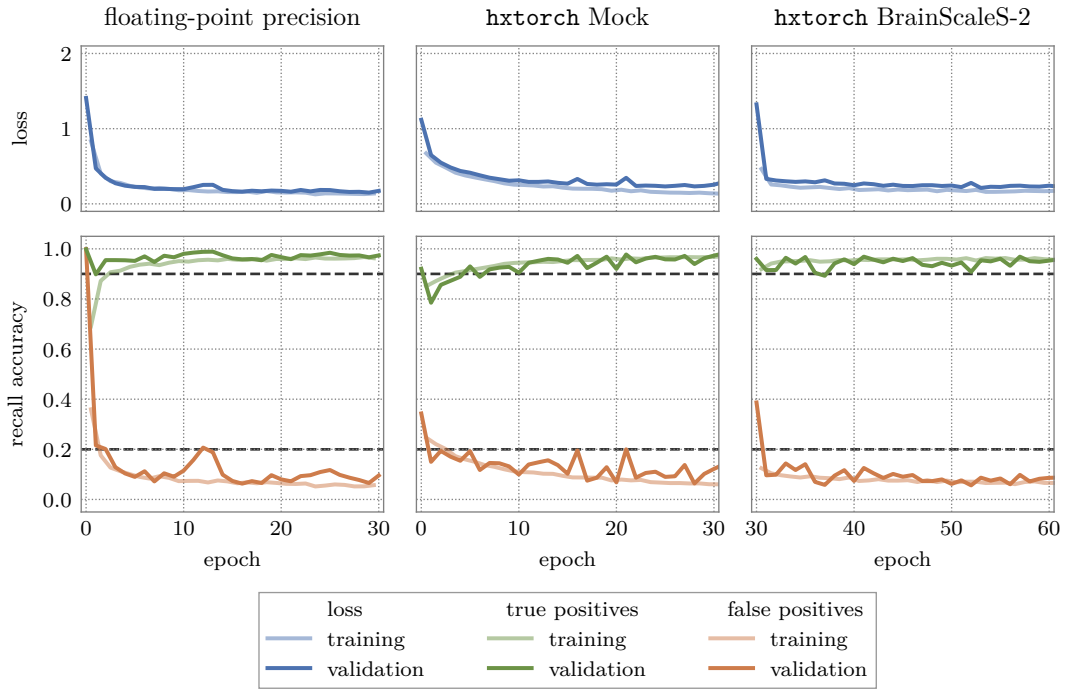

Figure 5.5: Training and validation metrics of the first proposed CNN model. The validation is performed on a separate test set (20 % split). The dashed lines in the lower plots represent the minimum requirements of the BMBF competition. The run with floating-point precision (on the left) is realized with conventional layers in PyTorch, using an additional 50 % dropout between the first and second layer to reduce overfitting. The simulation (center) achieves comparable results despite of quantized weights and statistical noise. After training in simulation it is continued on BrainScaleS-2 (right), after one more epoch a comparable accuracy is achieved. Thus the analog hardware is also capable of exceeding the requirements of the BMBF competition. No significant overfitting can be found both in simulation and using BrainScaleS-2.

It is hardly possible to transfer a model trained in simulation directly to the analog hardware without loss of accuracy, but less than one epoch of training with hardware in the loop is enough to almost reach the previous performance. This is due to fixed-pattern deviations of the analog circuitry, shown previously in figure 4.6, which generally distort the results. However, as the weights of the first layer are placed identically at different

locations, this layer can not compensate these deviations during training. Astonishingly, the network is still able to achieve good performance, with only the second and third layer actually compensating hardware distortions.

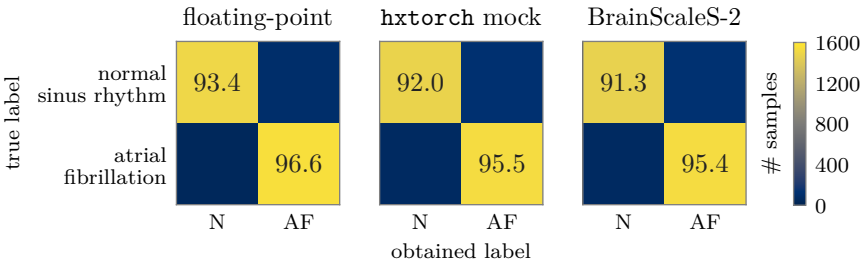

Figure 5.6: Confusion matrices and recall accuracies of the first convolutional neural network model presented in figure 3.5. All results significantly exceed the requirements of the BMBF competition, which correspond to recall accuracies of at least 80 % for normal sinus rhythm and 90 % for atrial fibrillation.

The final confusion matrices on the separate validation split are shown in figure 5.6. Trained with floating-point precision in PyTorch, the model achieves up to 96.6 % detection accuracy of atrial fibrillation at a false positive rate of only 6.6 %. The results of the mock mode and on BrainScaleS-2 are quite comparable. The analog hardware performs even slightly better than the simulation and achieves 95.7 % and 10.7 %, respectively.

As sketched in figure 5.4, all of the weights of the CNN fit at the same time on the two synapse arrays of 128×256 values each. The big advantage of this is that during inference the chip hardware does not need any reconfiguration. This avoids a lot of overhead, because all weights have to be sent to the chip and configured to the synapses array only once. The time for this configuration is not considered in the following, since the rules of the competition allow an initial configuration of the setup before the start of the energy measurement. To classify a single ECG recording with the first CNN model outlined, 648 multiplications with the convolutional kernel are performed in the first layer. As 18 of these one-dimensional convolution operations are combined and executed in parallel as described in section 4.2.2, only 36 vector-matrix multiplications are necessary for this task. Two further vector-matrix multiplications are needed for the subsequent two fully connected layers. However, the calculation of the last two layers can be executed simultaneously with the first layer of the next ECG recording. Therefore, the average classification time corresponds to the runtime of the first layer, i. e., the duration of 36 vector-matrix multiplications.

According to Weis (2020), BrainScaleS-2 can handle one vector-matrix multiplication every 5 µs, however, this represents the hardware limit and is not yet reached. Thus, it should be possible to classify an ECG trace with the first model every 180 µs. Since the power consumption of the BrainScaleS-2 setup is expected to be almost constant, the runtime is directly proportional to the required energy per classification. The BrainScaleS-2 chip itself has a power consumption of about 0.36 W, which includes both the analog core and digital communication (Weis, 2020). Therefore the average power consumption of the BrainScaleS-2 system needed for classification is about 65 µJ per ECG recording.

5.1.2.2 Optimizing the Runtime: The Second CNN Model

With the results presented in figure 5.6, the first CNN model surpasses significantly the requirements of the BMBF competition of 90 % detection rate and 20 % false positives. Therefore, it might be possible to use only a part of the ECG recordings for inference and still meet the minimum requirements. This is especially desirable, since the used length of the input data is directly proportional to the needed runtime and thus energy consumption. Figure 5.7 shows the effects of two approaches to reduce the required input length of the model: reduction of the pool width between the first and second layer and an increase of the stride in the first layer.

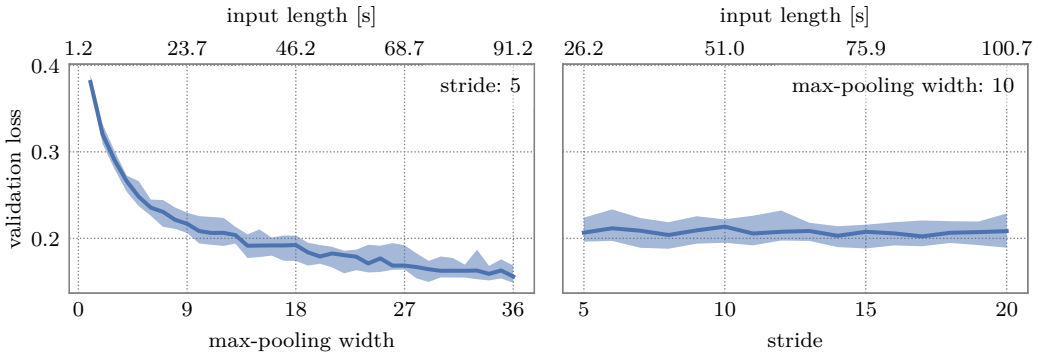

Figure 5.7: Sweep of max-pooling (left), and stride of the convolutional layer (right). Both are proportional to the input length used, which is shown in the upper part of the plots. Each combination is trained in PyTorch for 15 epochs with floating point accuracy. The blue curve shows the median of 40 runs, all other results are within the colored area. Even a loss of less than about 0.3 is sufficient to reliably reach the minimum accuracy requirements of the BMBF competition.

It is shown that pooling can be reduced significantly and therefore even smaller parts of the ECG recordings are sufficient to achieve the required accuracy. Already a pool width of 4, i. e., an input length of about 12 s, fulfills the minimum requirements. Surprisingly, increasing the stride of the first layer shows no significant effect on the result. Therefore, it is possible to reduce the pooling width between the layers even more while keeping the input length constant.

An optimized CNN model which takes these findings into account is shown in figure 5.8. The stride of the first layer is increased to 20 and the max-pooling is completely omitted. Due to these changes it operates on input lengths of 12 s, i. e., only 10 % of the data. With this proposed model, only two consecutive multiply accumulate operations must be performed on each synapse array to classify a single ECG recording. Since they run in parallel for successive recordings, effectively the integration time for two operations is needed, i. e., 10 µs per trace. This means that the BrainScaleS-2 system only requires an average of 3.6 µJ per classification, the energy consumption is reduced by a factor of 18 compared to the first CNN model.

An average runtime of 5 µs to 10 µs is also achieved by the STP method presented in

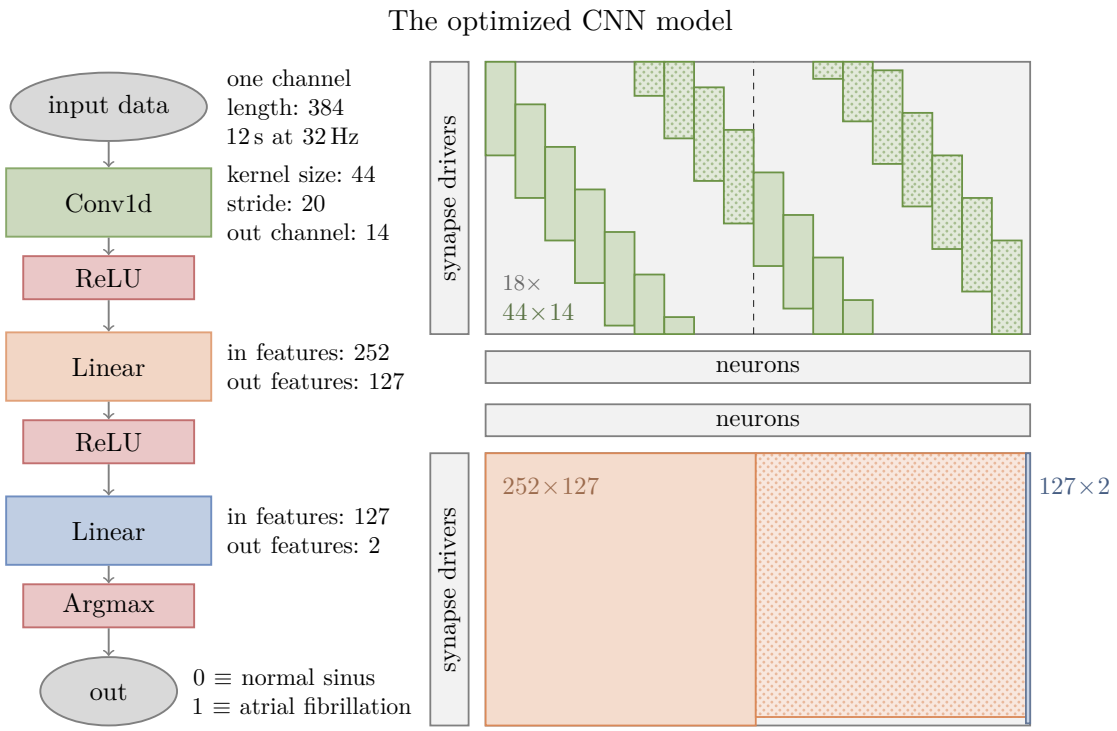

Figure 5.8: Layer structure (left) and on-chip arrangement (right) of the optimized convolutional neural network model that does not need any pooling between its layers. Similar to the dense layer (orange) in the lower synapse array, the convolutional layer (green) has to be split in order to process the input data in as few steps as possible. The two different textures of the weights (uniform and dotted) correspond to different label bits and are multiplied with different parts of the input vectors. In addition, two separate steps are necessary for this layer (separated by the dashed line), since only two independent input vectors are possible.

Weis (2020), which provides even better results so far. Unlike the latter, however, a CNN is not task specific, which is one of the requirements of the competition. In addition, a reliable beat detection is required. In contrast to the STP method, a sequential processing of the data is possible with short runtime, which significantly reduces the latency.

Another advantage of using only a short section of an ECG recording is the possibility to split the available recordings, which results in much more training samples. In addition, the accuracy can be significantly improved by sensible selection of the window used for inference. Figure 5.9 shows the achieved validation loss of a trained model for different starting points of the used part from the recording. It is noticeable that the model performs better at the end. One possible explanation is that the subjects still move after starting the recording, which causes physiological noise. Towards the end, the probability of less interference and a steady heartbeat is significantly increased.

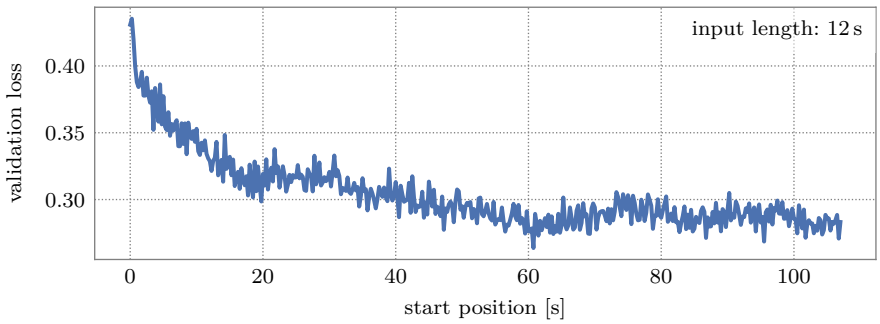

Figure 5.9: Sweep of the start position at inference with only 12 s input length. Towards the end of the ECG recordings the average loss is lower and therefore the precision is significantly higher than at the beginning. This can be explained by increased physiological noise directly after starting the recording. After about 20 s the subject usually seems to be in a resting position and disturbances due to muscle activity become less likely towards the end.

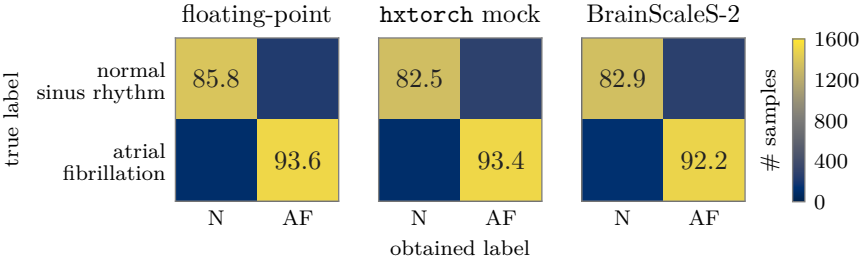

Figure 5.10: Confusion matrices and recall accuracies of the runtime optimized CNN model. For these results, a 12 s interval is used, starting at the 60$^{\text{th}}$ second of each ECG recording.

The confusion matrices for the optimized model are shown in figure 5.10. Although the accuracies of the first model are not achieved, it has a significantly lower runtime

and thus energy consumption and still significantly outperforms the requirements of the BMBF competition. 92.2 % of atrial fibrillation and 82.9 % of normal sinus rhythm can be correctly classified using the BrainScaleS-2 chip. The higher results with floating-point precision compared to the hardware indicate that the accuracy on BrainScaleS-2 can be further optimized by manually adjusting the hardware parameters. However, it is hardly possible to increase the parallelization of the operations in order to further optimize the runtime and thus the energy consumption during inference.

5.1.2.3 Universality of the Trained Model

The dataset of the PhysioNet Challenge 2017 presented in section 2.3.1.6 is used to evaluate whether a model trained on the BMBF competition data can be applied to other ECG recordings as well without further modifications. For this purpose, the entire dataset is filtered for recordings of at least 12 s in length and normal sinus rhythm or atrial fibrillation. The remaining 5917 recordings are transformed by trigonometric interpolation (see section 3.1.3) to 512 Hz sample rate and shortened to 12 s. As shown in listing 5.1, this can be achieved conveniently by using the preprocessing framework introduced in section 4.1. The format of this processed data corresponds to the shortened ECG recordings from the BMBF dataset (section 2.3.1.5) that are used to train the optimized CNN model, without being scaled or manipulated in any other way.

```
# load both training and test set:
loader = Physionet2017Loader(path)
dataset = loader.load_train() + loader.load_test()
# define the dataset transformation:
transform = Compose(
    Filter(lambda s: s.label in (Normal, AFib) and len(s) >= 12),
    Resample(512),  # resample to 512 Hz
    FixedLength(12),  # shorten to 12 seconds
)
dataset = transform(dataset)  # transform the dataset
```

Listing 5.1: Code example using the preprocessing framework presented in section 4.1 to transform the ECG recordings from the PhysioNet Challenge 2017 dataset into a compatible format for the trained model.

Despite the different imaging systems with different placement of the electrodes on the body, the model can still detect 92.1 % of arrhythmias with a false positive rate of 16.8 % in the PhysioNet dataset. This is below the 93.6 % and 14.2 % on the validation split of the BMBF dataset, but it shows that the developed model is general enough to be applied to other sources of ECG recordings without further training.

5.2 Human Activity Recognition

The universality of inference with a one-dimensional convolutional neural network model on the BrainScaleS-2 hardware is additionally demonstrated with an activity detection example. For this purpose, the Human Activity Recognition dataset presented in section 2.3.2 is used, which contains already denoised acceleration sensor data of a smartphone.

The model applied is presented in section 3.2.4. It corresponds to the CNN models used for the classification of ECG recordings apart from minor adaptations. The number of input channels and outputs is increased, the rest of the architecture remains the same. The model is trained on the training part of the dataset, the training metrics are shown in figure 5.11. The loss during validation is not significantly higher than the loss on the training dataset, there is no significant overfitting. After 10 epochs of training, an accuracy of almost 90 % is reached both on the training and on the validation dataset.

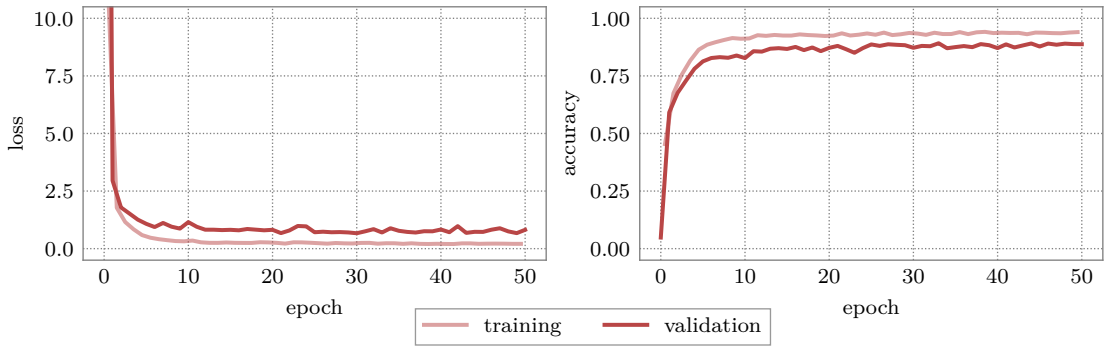

Figure 5.11: Training statistics for the Human Activity Recognition model in the `hxtorch` mock mode simulation. The model shows a smooth convergence with little overfitting. An overall classification accuracy of 88.2 % is achieved on the test set.

Figure 5.12 shows the resulting confusion matrices after three different training runs of 50 epochs, each. Training with floating-point precision in PyTorch achieves an overall accuracy of 89.7 % on the separate test set. This is significantly less than the 96 % achieved in Anguita et al. (2013). Nevertheless, the results are still rather impressive considering that the model architecture was originally designed for the classification of ECG recordings. The errors of the model are understandable: as already shown in figure 2.13, standing and sitting are both resting activities with almost identical orientation of the smartphone at the belt. The results of the quantized models with 88.2 % in the simulation and 88.8 % on the BrainScaleS-2 hardware are only slightly lower. It is shown again that the mock mode of `hxtorch` sufficiently describes the properties of the hardware and leads to comparable results.

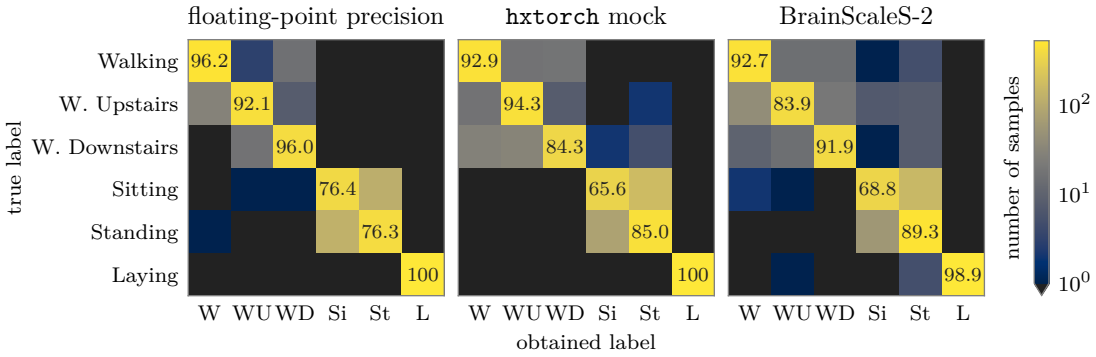

Figure 5.12: Confusion matrices of the trained Human Activity Recognition model on the separate test set. The numbers correspond to the recall accuracy of the corresponding class. There is little confusion between mobile and resting activities. As expected, sitting and standing are the least clearly distinguishable due to the similar orientation of the smartphone.

6 Discussion and Outlook

BrainScaleS-2 is capable of performing vector-matrix multiplications in its analog core, allowing for inference with artificial neural networks while promising good energy efficiency. In the presented work, networks are designed that are capable to classify ECG recordings on the chip but can also be generalized to other tasks like human activity recognition.

To preprocess ECG recordings, different methods are evaluated (section 3.1). This includes correction of the baseline and detection of the beat positions. A main challenge are strong distortions in the provided data, it is found that even existing peak detection algorithms are not able to compensate for them sufficiently. Nevertheless, through a combination of a discrete derivative and a prominence-based algorithm a beat detection method is realized. To combine, transform and compare ECG recordings from different sources, a new preprocessing framework is implemented. It is designed for the use in machine learning experiments, and its versatility and modularity also makes it suitable for other time-based datasets.

Different methods are presented to identify normal sinus rhythm and atrial fibrillation (section 3.2). It is shown that the mere position of the beats is sufficient to reliably distinguish these two classes. From a histogram of deviations between successive beat intervals, 93.6 % of the recordings with atrial fibrillation can be classified with a false positive rate of 9.3 %. This already fulfills the requirements of the BMBF competition, but is very task specific. Convolutional neural networks promise a higher versatility while requiring less preprocessing.

When designing a convolutional neural network model for the use with BrainScaleS-2, the focus is on ensuring that all weights fit together in the two 128×256 synapse arrays of the chip. This approach eliminates the need to reconfigure the weights during inference of an ECG sample, thus avoiding the significant overhead of transferring and configuring different weights. Based on this principle, multiple models are derived and presented (section 5.1.2). The first CNN model operates on the entire input data. Using the BrainScaleS-2 system, 95.4 % of recordings with atrial fibrillation can be correctly identified at a false positive rate of 8.7 %. With floating point accuracy even 96.6 % and 6.6 % are achieved. The model therefore clearly exceeds the minimum requirements of the BMBF competition of at least 90 % and at most 20 % false positives. For the execution of this first version 36 successive vector matrix multiplication operations are required. It is assumed that a single full-chip multiply accumulate operation takes 5 µs at a constant power consumption of about 0.36 mW (Weis, 2020), which results in 1.8 µJ per operation. This allows a batch of 500 ECG recordings to be classified using BrainScaleS-2 within 90 ms at an energy consumption of 32.4 mJ. Further investigations show that much smaller parts of the data are sufficient to achieve the required accuracy. A CNN model optimized for runtime still achieves an accuracy of 92.2 % on the analog hardware at a false positive

rate of 17.1 % and requires significantly fewer operations than the first model. This makes it possible to process 500 ECG recordings in only 5 ms and thus reduces the required energy to 1.8 mJ. However, the current hardware revision requires an external FPGA for memory access. Its power consumption is assumed to be in the range of a few watts and will therefore account for the majority of the energy consumption. Since the evaluation basis of the BMBF competition is the absolute energy consumption of the system (see chapter 1), the CNN models are optimized for low runtime to realize a short power-on time.

The initialization of weights and the correct scaling of gradients in the backward pass, initially thought to be negligible, proved to be decisive for successful training with multiple layers on the BrainScaleS-2 chip. Both are integrated into `hxtorch` and now handled automatically (section 4.2). Additionally, the convolution operation is further optimized by placing logical weights several times side by side in the physical weight arrays of the chip. The typically small weight matrices allow a high degree of parallelization and a significant reduction of the runtime of this operation.

`hxtorch` is also extended by an additional mock mode (section 4.2.3). It allows training of compatible models without requiring physical access to the BrainScaleS-2 system by simulating the statistical noise and value ranges of the BrainScaleS-2 chip. It turns out that, especially with low inputs, the linear approximation reaches its limits. Despite of this, fixed deviations of the individual circuits and non-linearities are not simulated in favor of a short runtime, but it is shown that the models trained in simulation could adapt to the individual deviations of the hardware within a short time by training with hardware in the loop.

It remains to be evaluated to what extent the used preprocessing affects the energy consumption. A further reduction at the cost of a larger convolutional neural network might be even more energy efficient than the current proposal. But this would probably require a reconfiguration of the weights on the chip, which again increases the required runtime and energy cost for weight transfer. Since the final hardware setup for the use in the competition is still in production, such experiments are pending. Beside the FPGA, the memory and the power supply circuits, it will also contain measurement components that enable such evaluations.

Currently, it is still quite complex to adapt an existing model trained with floating point accuracy and run it on BrainScaleS-2. This is mainly due to the fixed output range of the chip. The factor between input and output is currently very small and can be too low for sparse input signals or small filter sizes, such as those used in image recognition. This can be individually compensated in each layer to a certain extent by sending the input several times or by using multiple synapses in parallel, but requires a lot of manual adjustment in the current software state. A possible solution for the automatic adjustment of the gain is based on similar ideas as the derivation of the initialization distributions in section 2.1.4: with the aim of keeping the amplitude variation of the input and output signals constant throughout the network, the required gain could be determined and adjusted for each layer during training. An increase of the gain could also be achieved by reducing the membrane capacitance, but this in turn increases the noise and causes a

faster decay of the results. Another possibility would be to increase the time constant of the synaptic inputs, thus allowing more charge to flow onto the membrane in the longer period of time. However, this may cause saturation effects that affect the linearity of the operation. The challenge here is that, in addition to increasing the gain, other properties are also influenced, which can lead to unexpected effects. In addition, BrainScaleS-2 is very versatile in its application, so many parameters of the circuits can be individually adapted to the needs of the experiments. This also applies to the parameters of the multiplication mode, so the properties listed in table 2.1 represent only one of numerous possibilities (cf. Weis, 2020).

In the current implementation, the mock mode of `hxtorch` only assumes statistical noise and a linear relation between input and output, since this is already sufficient to achieve a comparable behavior to the chip during training. But extending it with more details of BrainScaleS-2 like fixed pattern deviations and non-linearities at low inputs could be desirable. This allows models that take advantage of these effects to be trained and investigated with sufficient detail, even in simulation.

Furthermore, it remains to be evaluated to what extent other neural network architectures, such as long short-term memory and attention, can be accelerated energy-efficiently with BrainScaleS-2. It would also be conceivable to perform the calculation of the gradient on-chip as well. Another outstanding feature of BrainScaleS-2, which was not used in this thesis, is the possibility to combine spiking and conventional artificial neural networks on a single substrate. A possible application would be the detection of features using a convolutional neural network which are then classified by a spiking neural network. This could be especially advantageous in typical applications of edge inference at high data rates, such as video processing with low power requirements. Maybe we might even see an autonomously flying BrainScaleS-2 chip in the future?

Bibliography

Anan, Tsuyoshi, Kenji Sunagawa, Haruo Araki, and Motoomi Nakamura (1990). "Arrhythmia analysis by successive RR plotting". In: *Journal of Electrocardiology* 23.3, pp. 243–248. ISSN: 0022-0736. DOI: 10.1016/0022-0736(90)90163-V.

Ananthanarayanan, G., P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha (2017). "Real-Time Video Analytics: The Killer App for Edge Computing". In: *Computer* 50.10, pp. 58–67. DOI: 10.1109/MC.2017.3641638.

Anguita, Davide, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz (2013). "A public domain dataset for human activity recognition using smartphones." In: *Esann*.

ARM (2020a). *Ethos-U55 Product Brief*. URL: https://armkeil.blob.core.windows.net/developer/Files/pdf/ML%20on%20Arm/Arm_Ethos_U55_Product_Brief_v4.pdf (visited on 10/14/2020).

– (2020b). *Ethos-U65 Product Brief*. URL: https://armkeil.blob.core.windows.net/developer/Files/pdf/arm-ethos-u65-product-brief-v2.pdf (visited on 10/22/2020).

Bishop, Christopher M. (2007). *Pattern recognition and machine learning*. 5. (corr. print.) Information science and statistics. New York [u.a.]: Springer, XX, 738 S.

Brandl, Georg (2020). *Sphinx Documentation Release 4.0.0+*. URL: https://www.sphinx-doc.org/_/downloads/en/master/pdf/ (visited on 10/19/2020).

Bundesministerium für Bildung und Forschung (BMBF) (2019). *Bekanntmachung: Richtlinie zur Förderung des Pilotinnovationswettbewerbs "Energieeffizientes KI-System"*. German. URL: https://www.bmbf.de/foerderungen/bekanntmachung-2371.html (visited on 09/07/2020).

Chung, Mina K, Lee L Eckhardt, Lin Y Chen, Haitham M Ahmed, Rakesh Gopinathannair, José A Joglar, Peter A Noseworthy, Quinn R Pack, Prashanthan Sanders, and Kevin M Trulock (2020). "Lifestyle and Risk Factor Modification for Reduction of Atrial Fibrillation: A Scientific Statement From the American Heart Association". In: *Circulation (New York, N.Y.)* 141.16, e750–e772. DOI: 10.1161/CIR.0000000000000748.

Clifford, Gari D, Chengyu Liu, Benjamin Moody, Li-Wei H Lehman, Ikaro Silva, Qiao Li, A E Johnson, and Roger G Mark (2017). "AF Classification from a Short Single Lead ECG Recording: the PhysioNet/Computing in Cardiology Challenge 2017". In: *Computing in cardiology* 44. ISSN: 2325-8861 and 2325-887X.

Coral, Google (2020). *Edge TPU performance benchmarks*. URL: https://coral.ai/docs/edgetpu/benchmarks (visited on 10/14/2020).

Cramer, Benjamin, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Johannes Schemmel, and Friedemann Zenke (2020). "Training spiking multi-layer

networks with surrogate gradients on an analog neuromorphic substrate". In: arXiv: 2006.07239 [cs.NE].

Cramer, Benjamin, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke (2019). "The Heidelberg spiking datasets for the systematic evaluation of spiking neural networks". In: arXiv: 1910.07407 [cs.NE].

Czierlinski, Milena (2020). "PyNN for BrainScaleS-2". Bachelorarbeit. Universität Heidelberg.

Davison, Andrew, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger (2009). "PyNN: a common interface for neuronal network simulators". In: *Frontiers in Neuroinformatics* 2, p. 11. DOI: 10.3389/neuro.11.011.2008.

Deng, J., W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei (June 2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

Elgendi, Mohamed, Björn Eskofier, Socrates Dokos, and Derek Abbott (Jan. 2014). "Revisiting QRS Detection Methodologies for Portable, Wearable, Battery-Operated, and Wireless ECG Systems". In: *PLOS ONE* 9.1, pp. 1–18. DOI: 10.1371/journal.pone.0084018.

Fuster, Valentin, Lars E. Rydén, David S. Cannom, Harry J. Crijns, Anne B. Curtis, Kenneth A. Ellenbogen, Jonathan L. Halperin, Jean-Yves Le Heuzey, G. Neal Kay, James E. Lowe, S. Bertil Olsson, Eric N. Prystowsky, Juan Luis Tamargo, and Samuel Wann (2006). "ACC/AHA/ESC 2006 Guidelines for the Management of Patients With Atrial Fibrillation—Executive Summary: A Report of the American College of Cardiology/American Heart Association Task Force on Practice Guidelines and the European Society of Cardiology Committee for Practice Guidelines (Writing Committee to Revise the 2001 Guidelines for the Management of Patients With Atrial Fibrillation) Developed in Collaboration With the European Heart Rhythm Association and the Heart Rhythm Society". In: *Journal of the American College of Cardiology* 48.4, pp. 854–906. DOI: 10.1016/j.jacc.2006.07.009.

George, Boby and Laurie Williams (2004). "A structured experiment of test-driven development". In: *Information and Software Technology* 46.5. Special Issue on Software Engineering, Applications, Practices and Tools from the ACM Symposium on Applied Computing 2003, pp. 337–342. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2003.09.011.

Glorot, Xavier and Yoshua Bengio (May 2010). "Understanding the difficulty of training deep feedforward neural networks". In: ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: JMLR Workshop and Conference Proceedings, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

Greff, Klaus, Aaron Klein, Martin Chovanec, Frank Hutter, and Jürgen Schmidhuber (2017). "The Sacred Infrastructure for Computational Research". In: *Proceedings of the 16th Python in Science Conference*. Ed. by Katy Huff, David Lippa, Dillon Niederhut, and M Pacer, pp. 49–56. DOI: 10.25080/shinma-7f4c6e7-008.

Guyton, A.C. and J.E. Hall (2006). *Textbook of Medical Physiology*. Elsevier Saunders. ISBN: 9780721602400.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning : Data Mining, Inference, and Prediction, Second Edition.* Vol. Second edition, corrected 7th printing. Springer Series in Statistics. Springer. ISBN: 9780387848570.

He, K., X. Zhang, S. Ren, and J. Sun (Dec. 2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

Hochreiter, Sepp and Jürgen Schmidhuber (Dec. 1997). "Long Short-term Memory". In: *Neural computation* 9, pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

Kay, Alan C. (1993). "The Early History of Smalltalk". In: *The Second ACM SIGPLAN Conference on History of Programming Languages*. HOPL-II. Cambridge, Massachusetts, USA: Association for Computing Machinery, pp. 69–95. ISBN: 0897915704. DOI: 10.1145/154766.155364.

Koehler, Friedrich, Sebastian Winkler, Michael Schieber, Udo Sechtem, Karl Stangl, Michael Böhm, Herbert Boll, Gert Baumann, Marcus Honold, Kerstin Koehler, Goetz Gelbrich, Bridget-Anne Kirwan, and Stefan D. Anker (2011). "Impact of Remote Telemedical Management on Mortality and Hospitalizations in Ambulatory Patients With Chronic Heart Failure". In: *Circulation* 123.17, pp. 1873–1880. DOI: 10.1161/CIRCULATIONAHA.111.018473.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 1097–1105.

Lee, Y., P. Tsung, and M. Wu (2018). "Techology trend of edge AI". In: *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–2. DOI: 10.1109/VLSI-DAT.2018.8373244.

Leshno, Moshe, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken (1993). "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural networks* 6.6, pp. 861–867. ISSN: 0893-6080 and 1879-2782.

Linde, Denise van der, Elisabeth EM Konings, Maarten A Slager, Maarten Witsenburg, Willem A Helbing, Johanna JM Takkenberg, and Jolien W Roos-Hesselink (2011). "Birth prevalence of congenital heart disease worldwide: a systematic review and meta-analysis". In: *Journal of the American College of Cardiology* 58.21, pp. 2241–2247.

Liu, Z., J. Wang, and B. Liu (2011). "ECG Signal Denoising Based on Morphological Filtering". In: *2011 5th International Conference on Bioinformatics and Biomedical Engineering*, pp. 1–4.

Lorenz, Edward N. (Mar. 1963). "Deterministic Nonperiodic Flow". In: *Journal of the Atmospheric Sciences* 20.2, pp. 130–141. ISSN: 0022-4928. DOI: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

Madeiro, João P.V., Paulo C. Cortez, João A.L. Marques, Carlos R.V. Seisdedos, and Carlos R.M.R. Sobrinho (2012). "An innovative approach of QRS segmentation based on first-derivative, Hilbert and Wavelet Transforms". In: *Medical Engineering & Physics* 34.9, pp. 1236–1246. DOI: 10.1016/j.medengphy.2011.12.011.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Masters, Dominic and Carlo Luschi (2018). *Revisiting Small Batch Training for Deep Neural Networks*. arXiv: 1804.07612 [cs.LG].

McIntosh, Shane, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan (2014). "The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects". In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: Association for Computing Machinery, pp. 192–201. ISBN: 9781450328630. DOI: 10.1145/2597073.2597076.

Merenda, Massimo, Carlo Porcaro, and Demetrio Iero (Apr. 2020). "Edge Machine Learning for AI-Enabled IoT Devices: A Review". In: *Sensors* 20.9, p. 2533. ISSN: 1424-8220. DOI: 10.3390/s20092533.

Moody, G. B. and R. G. Mark (2001). "The impact of the MIT-BIH Arrhythmia Database". In: *IEEE Engineering in Medicine and Biology Magazine* 20.3, pp. 45–50. DOI: 10.1109/51.932724.

Pan, Jiapu and Willis J Tompkins (1985). "A Real-Time QRS Detection Algorithm". In: *IEEE transactions on biomedical engineering* BME-32.3, pp. 230–236. DOI: 10.1109/TBME.1985.325532.

Pandit, Diptangshu, Li Zhang, Chengyu Liu, Samiran Chattopadhyay, Nauman Aslam, and Chee Peng Lim (2017). "A lightweight QRS detector for single lead ECG signals using a max-min difference algorithm". In: *Computer Methods and Programs in Biomedicine* 144, pp. 61–75. ISSN: 0169-2607. DOI: 10.1016/j.cmpb.2017.02.028.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: arXiv: 1912.01703 [cs.LG].

Patchett, Nicholas (2015). *Derivation of the limb leads*. Npatchett, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Limb_leads_of_EKG.png.

Python Code Quality Authority (2020a). *Pycodestyle Documentation*. URL: https://pycodestyle.pycqa.org (visited on 09/30/2020).

– (2020b). *Pylint User Manual*. URL: http://pylint.pycqa.org (visited on 09/30/2020).

Rajpurkar, Pranav, Awni Y. Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y. Ng (2017). *Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks*. arXiv: 1707.01836 [cs.CV].

Rangayyan, Rangaraj M. (2002). *Biomedical Signal Analysis - A Case-Study Approach*. New York: Wiley. ISBN: 978-0-471-20811-2.

Rodríguez, R, A Mexicano, J Bila, S Cervantes, and R Ponce (2015). "Feature Extraction of Electrocardiogram Signals by Applying Adaptive Threshold and Principal Component

Analysis". In: *Journal of applied research and technology* 13.2, pp. 261–269. DOI: 10.1016/j.jart.2015.06.008.

Rossum, Guido van, Jukka Lehtosalo, and Łukasz Langa (Dec. 2014). *Type Hints*. PEP 484. URL: https://www.python.org/dev/peps/pep-0484/.

Rossum, Guido van, Barry Warsaw, and Nick Coghlan (2001). *Style Guide for Python Code*. PEP 8. URL: https://www.python.org/dev/peps/pep-0008/.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, pp. 533–536.

Schemmel, Johannes, Sebastian Billaudelle, Phillip Dauer, and Johannes Weis (2020). *Accelerated Analog Neuromorphic Computing*. arXiv: 2003.11996 [cs.NE].

Schemmel, Johannes, Steffen Hohmann, Karlheinz Meier, and Felix Schürmann (Feb. 2004). "A Mixed-Mode Analog Neural Network Using Current-Steering Synapses: Special Issue on Current Mode Circuit Techniques". In: *Analog Integrated Circuits and Signal Processing* 38. DOI: 10.1023/B:ALOG.0000011170.92377.6e.

Schmitt, Sebastian, Johann Klahn, Guillaume Bellec, Andreas Grubl, Maurice Guttler, Andreas Hartel, Stephan Hartmann, Dan Husmann, Kai Husmann, Sebastian Jeltsch, and et al. (May 2017). "Neuromorphic hardware in the loop: Training a deep spiking network on the BrainScaleS wafer-scale system". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. DOI: 10.1109/ijcnn.2017.7966125.

Spilger, Philipp, Eric Müller, Arne Emmel, Aron Leibfried, Christian Mauch, Christian Pehle, Johannes Weis, Oliver Breitwieser, Sebastian Billaudelle, Sebastian Schmitt, Timo C. Wunderlich, Yannik Stradmann, and Johannes Schemmel (June 2020). "hxtorch: PyTorch for BrainScaleS-2 — Perceptrons on Analog Neuromorphic Hardware". In: *Proceedings of the Workshop on IoT, Edge, and Mobile for Embedded Machine Learning (ITEM)/ECML-PKDD 2020 (accepted)*. arXiv: 2006.13138 [cs.NE].

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller (2014). *Striving for Simplicity: The All Convolutional Net*. arXiv: 1412.6806 [cs.LG].

Übeyli, Elif Derya (2008). "Implementing wavelet transform/mixture of experts network for analysis of electrocardiogram beats". In: *Expert Systems* 25.2, pp. 150–162. DOI: 10.1111/j.1468-0394.2008.00444.x.

Verleysen, M., P. Thissen, J. -. Voz, and J. Madrenas (1994). "An analog processor architecture for a neural network classifier". In: *IEEE Micro* 14.3, pp. 16–28. DOI: 10.1109/40.285221.

Virtanen, Pauli et al. (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Weis, Johannes (Sept. 2020). "Inference with Artificial Neural Networks on Neuromorphic Hardware". Master's thesis. Universität Heidelberg.

Weis, Johannes, Philipp Spilger, Sebastian Billaudelle, Yannik Stradmann, Arne Emmel, Eric Müller, Oliver Breitwieser, Andreas Grübl, Joscha Ilmberger, Vitali Karasenko, Mitja Kleider, Christian Mauch, Korbinian Schreiber, and Johannes Schemmel (June 2020). "Inference with Artificial Neural Networks on Analog Neuromorphic Hardware". In: *Proceedings of the Workshop on IoT, Edge, and Mobile for Embedded Machine Learning (ITEM)/ECML-PKDD 2020 (accepted)*. arXiv: 2006.13177 [cs.NE].

Wyse, D.G., A.L. Waldo, J.P. DiMarco, M.J. Domanski, Y. Rosenberg, E.B. Schron, J.C. Kellen, H.L. Greene, M.C. Mickel, J.E. Dalquist, and S.D. Corley (2002). "A Comparison of Rate Control and Rhythm Control in Patients with Atrial Fibrillation". In: *New England Journal of Medicine* 347.23, pp. 1825–1833. DOI: 10.1056/NEJMoa021328.

Yamaguchi, Masatoshi, Goki Iwamoto, Hakaru Tamukoh, and Takashi Morie (2019). *An Energy-efficient Time-domain Analog VLSI Neural Network Processor Based on a Pulse-width Modulation Approach*. arXiv: 1902.07707 [cs.ET].

Zhao, Jianxin, Richard Mortier, Jon Crowcroft, and Liang Wang (2018). "Privacy-preserving machine learning based data analytics on edge devices". In: *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 341–346.

Zhou, Hai-Ying, Kun-Mean Hou, and De-Cheng Zuo (2009). "Real-Time Automatic ECG Diagnosis Method Dedicated to Pervasive Cardiac Care". In: *Wireless sensor network* 1.4, pp. 276–283. DOI: 10.4236/wsn.2009.14034.

The work carried out in this Master's Thesis used systems that received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements Nos. 785907 and 945539 (Human Brain Project, HBP).

Statement of Originality – Erklärung

I certify that this thesis, and the research to which it refers, are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, November 2, 2020