Decoding EEG Brain Signals using Recurrent Neural Networks

MASTER THESIS

submitted by
Juri Fedjaev

born 25.11.1991

Leonrodstrasse 72
80636 München
Tel.: 0176 43338743

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universität München

Prof. Dr. sc. nat. Jörg Conradt

Supervisor:	Dipl.-Ing. Zied Tayeb
Co-Supervisors:	Dr. Christoph Richter
	M.Sc. Lukas Everding
	Dr. Nejla Ghaboosi

Start:	06.06.2017
Intermediate Report:	20.09.2017
Final Submission:	12.12.2017

Technische Universität München

Neurowissenschaftliche Systemtheorie

MASTER THESIS

For Juri Fedjaev: Mat. -N.03628226, Electrical Engineering Department

Decoding of EEG Brain Signals Using Recurrent Neural Networks

Problem description:

Motor Imagery Electroencephalography (MI-EEG) plays an important role in brain machine interface (BMI) especially for rehabilitation robotics. However, it has the characteristics of nonlinear, non-stationary and time-varying sensitivity. Therefore, the key problem of BMI system is how to identify the most stable/invariant features from EEG signals and how to develop an adaptive algorithm to decode such non-stationary and very noisy signals with high accuracy and low power consumption. Previous attempts to analyze EEG signals have focused on well-characterized sensorimotor data but the BMI field seems to have stagnated in improving motor decoding using this method and deep learning algorithms [1] can present a promising solution to decode those signals without requiring hand-crafted features. As EEG is time series, it should be tremendously helpful to consider the time sequence information of the signal. The Recurrent Neural Networks (RNNs) such as Elman RNN (vanilla RNN model) or LSTM have recently emerged as an effective model in a wide variety of applications that involve sequential data such as speech recognition, handwriting recognition and natural language processing. However, RNN have been not widely used in the field of EEG [2].

The main objective of this thesis is to develop a recurrent neural network algorithm [2] to decode EEG brain signals during four motor imagery movements (left, right, both hands and rest) and to train it offline on CPU or GPU using **Theano** packages. Optionally, as on GPUs or CPUs a deep neural network requires a substantial amount of electrical power and computational resources, it would be desirable to convert the developed algorithm to Spiking neural network for the highly power-efficient TrueNorth Neuromorphic hardware [3]. With less than **70 mW** and its compact size, the IBM's chip presents an ideal platform for assistive patient devices, such as brain-machine interfaces and neuroprosthesis.

Tasks:

This master thesis project requires the student to:

- Review EEG decoding literature
- Develop and train a recurrent neural network for EEG decoding on CPU or GPU using Theano
- Compare the results obtained with LSTM implemented on [4] and with state-of-the art-results in literature.

Optional task:

- Implementation on **TrueNorth IBM's chip** by converting the developed RNN in a Spiking neural network algorithm
- Test and validate in a real-time scenario

Bibliography:

[1] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, "*Deep learning*", Nature 521,436–444, 2015

[2] "*Electroencephalogram classification by forecasting with recurrent neural networks"*, Department of Computer Science, Colorado State University, Master thesis, 2011

[3] Peter U. Diehl, Guido Zarrellay, Andrew Cassidyz, Bruno U. Pedronix and Emre Neftci, "*Conversion of Artificial Recurrent Neural Networks to Spiking Neural Networks for Low-power Neuromorphic Hardware*", arXiv:1601.04187v1 [cs.NE] 16 Jan 2016

[4] Mingai Li, Meng Zhang, Xinyong Luo and Jinfu Yang, "*Combined Long Short-Term Memory based Network employing wavelet coefficients for MI-EEG recognition*", International Conference on Mechatronics and Automation China, 2016

Supervisor: Dipl.-Ing. Zied Tayeb
Co-supervisors: Dr. Christoph Richter / M.Sc. Lukas Everding
Start: 05.06.2017
Intermediate Report: 04.09.2017
Delivery: 05.12.2017

(Jörg Conradt)
Professor

Abstract

Brain-computer interfaces (BCIs) based on electroencephalography (EEG) enable direct communication between humans and computers by analyzing brain activity. Specifically, modern BCIs are capable of translating imagined movements into real-life control signals, e.g., to actuate a robotic arm or prosthesis. This type of BCI is already used in rehabilitation robotics and provides an alternative communication channel for patients suffering from amyotrophic lateral sclerosis or severe spinal cord injury. Current state-of-the-art methods are based on traditional machine learning, which involves the identification of discriminative features. This is a challenging task due to the non-linear, non-stationary and time-varying characteristics of EEG signals, which led to stagnating progress in classification performance. Deep learning alleviates the efforts for manual feature engineering through end-to-end decoding, which potentially presents a promising solution for EEG signal classification.

This thesis investigates how deep learning models such as long short-term memory (LSTM) and convolutional neural networks (CNN) perform on the task of decoding motor imagery movements from EEG signals. For this task, both a LSTM and a CNN model are developed using the latest advances in deep learning, such as batch normalization, dropout and cropped training strategies for data augmentation.

Evaluation is performed on a novel EEG dataset consisting of 20 healthy subjects. The LSTM model reaches the state-of-the-art performance of support vector machines with a cross-validated accuracy of 66.20%. The CNN model that employs a time-frequency transformation in its first layer outperforms the LSTM model and reaches a mean accuracy of 84.23%. This shows that deep learning approaches deliver competitive performance without the need for hand-crafted features, enabling end-to-end classification.

I would like to thank my main supervisor Zied Tayeb, and both my co-supervisors Lukas Everding and Christoph Richter for guiding and inspiring me during this thesis. Special thanks go to Nejla Ghaboosi for being my mentor in deep learning techniques.

Contents

Chapter 1

Introduction

People with severe neuromuscular disorders, such as late-stage amyotrophic sclerosis (ALS) and those paralyzed from higher level spinal cord injury are unable to actuate any of their muscles. Communication with the outside world is therefore problematic for the suffering people. Cognitive and sensory body functions, however, are often only minimally affected. Therefore, an electroencephalogram (EEG)-based communication which does not require any neuromuscular control is considered to be particularly helpful to enhance the disabled's quality of life by increasing their independence [1].

This work focuses on EEG-based brain-computer interfaces (BCIs) for decoding motor imagery left- and right-hand movements to control a robotic arm or prosthesis, contributing to the advancement in rehabilitation robotics. The classification of motor imagery EEG is a challenging task. Due to the low signal-to-noise ratio and non-linear, time-varying characteristics of the signals, one of the key problems of BCI design is the identification of discriminative features. While previous work focusing on the well-known motor imagery patterns has achieved impressive results [2, 3], the progress in BCI performance has been stagnating in the recent years. Novel approaches in the field of deep learning present a promising solution to decode EEG signals without the need for creating hand-crafted features [4].

This work aims to investigate the potential of deep learning methods to classify binary motor imagery movements in EEG signals. Investigated approaches include the long short-term memory (LSTM) recurrent neural networks (RNNs) as well as convolutional neural networks (CNNs) (described in Sections 2.3.2 and 2.3.4, respectively). While RNN-based models have reached state-of-the-art performance in sequential data processing tasks, such as natural language processing [5], CNNs outperform previous methods in computer vision and speech recognition [6]. Given that neural activity measured by EEG can be regarded as highly dynamic, non-linear time series data, RNNs appear to be the tool of choice for modeling the temporal dynamics of brain activity. However, RNNs have not yet been widely used in the

field of EEG classification [7].

In the following, this thesis first describes the technical background of BCIs, including the biological characteristics and acquisition of EEG signals. Then, a brief introduction to the concepts of deep learning and related work in the area of BCI research is presented. Finally, the implementation of the proposed LSTM and CNN models is explained and evaluated on data recorded at NST.

Chapter 2

Technical Background

This chapter first introduces the principles and characteristics of EEG-based brain-computer-interfaces (BCIs) in general. Then, the underlying processes for oscillatory brain signals measured by EEG and a common practice EEG recording protocol are explained. Furthermore, basic concepts of neural networks and deep learning are described.

2.1 EEG-based Brain-Computer Interface

A BCI is a platform for communication between a human being and a machine that is based on brain signals, bypassing the need of neuromuscular involvement. There are both *invasive* and *non-invasive* methods of monitoring the brain's neural activity for the use in a BCI, whereby this work focuses on non-invasive electroencephalography (EEG)-based interfaces. A BCI typically aims to estimate its user's mental state or intent from the monitored signals and translate it into a physical action, e.g., to control a neural prosthesis. In general, a BCI relies on five main components: data collection or recording to retrieve raw neural activity, signal (pre-) processing to eliminate undesired noise, feature extraction to obtain abstract insights from the recording, classification to interpret the extracted features and decide on the user's intent, and a feedback or output stage that either provides feedback of the decision made by the system or translates the brain signals into physical movements of a robotic system [8].

BCIs can be categorized into two types based on the monitored EEG feature, i.e., *event-related* or *oscillatory* brain activity. *Event-related potential* (ERP)-based BCIs, also called *reactive BCIs*, rely on external time-locked stimuli. The stimuli provoke temporally correlated waveforms in the EEG signal that are well-characterized. ERP-based BCIs in general are relatively robust across subjects when compared to BCIs based on oscillatory processes [9]. BCIs based on oscillatory processes are also called *active BCIs*, because they do not rely on external stimuli, are not time-locked, and are able to translate voluntary, self-paced mental tasks into physical control [10].

Active BCIs are in general more challenging to design due to a lower signal-to-noise ratio (SNR) and strong variability across different subjects.
The underlying brain processes of oscillatory EEG features are briefly described in the following section.

2.1.1 Sensory Motor Rhythms and Motor Imagery

The EEG features of interest in this work are *sensory motor rhythms* (SMR). SMR are a common control signal for active BCIs because of their recognizable and stable patterns in the noisy EEG recordings. Characteristics of SMR can be described by the desynchronization of neural activity over the sensorimotor cortex in α- (8 Hz - 13 Hz) and β- (12 Hz - 25 Hz) frequency bands during a physical or imagined movement of limbs.

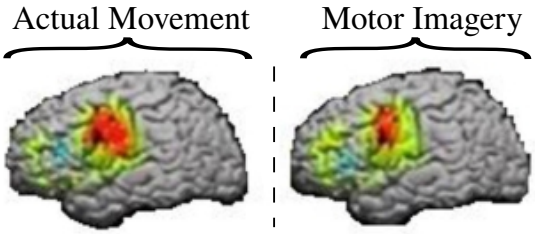

Figure 2.1: Visualization of activated brain regions during motor execution and imagery recorded by fMRI. Illustration adapted from [11].

SMR can be induced induced by the mental rehearsal of a physical task. This mental strategy is also refereed to as *motor imagery* (MI) [12]. On a neurophysiological level, similar brain regions are activated during motor execution and motor imagery, however, the performance is blocked at a corticospinal level. Studies based on fMRI showed similar activation patterns during motor imagery and actual movement execution [11], which can be seen in Figure 2.1. For operating an active BCI, motor imagery has proven its capability as an efficient mental strategy.

2.1.2 EEG Data Acquisition

The first stage in a BCI is the essential task of data collection. In order to systematically accumulate data for training MI-based decoding systems, de-facto standardized recording protocols have been designed.

EEG Recording Protocol for Motor Imagery

As an example for a typical EEG experimental paradigm for recording motor imagery, the *Graz B* dataset is explained [13]. In the data set three bipolar electrodes (*C3, C4* and *Cz*) are recorded with a sampling frequency of 250 Hz. The cue-based experimental paradigm is shown in Figure 2.2.

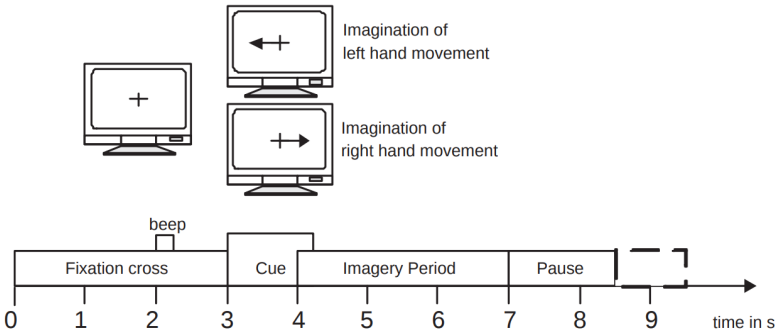

Figure 2.2: Typical EEG Recording Protocol (taken from Graz BCI Competition 2008, *data set B* [3])

The paradigm consists of two motor imagery classes for left and right hand movements. Each trial starts with a fixation cross following an acoustic warning signal at $t = 2$ s. At $t = 3$s a visual cue is presented for the duration of 1.25s. Subsequently, the participant was instructed to imagine the corresponding movement over a period of 4 seconds. A short break of 1.5 s followed the trial. Additionally, a randomized break of up to 1 second followed the previous break to avoid adaption of the participant to the protocol. The data set contains a total of 9 subjects with 120 trials each. After executing the motor imagery, for example for four seconds, a rest period and the preparation for the next trial follow. Since motor imagery is very exhausting, no more than 120 trials can be conducted without loss of quality. Electrooculography (EOG) was also recorded in the Graz B dataset, which further improves the data quality because the interference of eye movements can be removed through independent component analysis (ICA). Furthermore, trials that are strongly affected by muscular noise can be marked as *bad trials* and excluded from further analysis.

Electrode Placement

For the positioning of the EEG electrodes on the scalp an internationally recognized topology called the 10-20 EEG system is widely used (see Figure 2.3 for illustration). It was developed to ensure reproducibility and comparability of results by standardizing the electrode positions. The system is based on the relationship between the location of an electrode and the underlying area of the cerebral cortex. The "10" and "20" refer to the fact that the actual distances between adjacent electrodes are either 10% or 20% of the total front-back or right-left distance of the skull [14]. Previous studies show that electrode positions C3, Cz and C4 are suitable for recording characteristic motor imagery signals as they are directly covering part of the sensorimotor cortex.

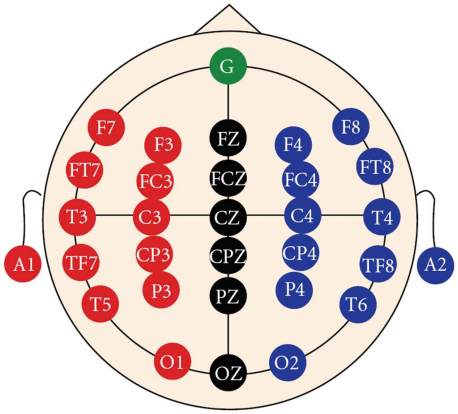

Figure 2.3: International standard 10-20 electrode placement system.

2.2 Artificial Neural Networks

This section introduces the concept of artificial neural networks and deep learning.

In the area of computer science and engineering, artificial neural networks (ANNs), or simply neural networks (NNs), are information processing systems inspired by the biological neural networks that constitute our brain [15]. A NN is a network of interconnected nodes (also known as *neurons*) that serve as simple processing units. Each neuron communicates with one or many other neurons using weighted connections, comparable to synapses in biological systems. When comparing ANNs with biological neural networks, both resemble each other in the way that both acquire knowledge in a learning process which is stored in the connections between the neurons [16].
Neural networks find their use in a wide variety of tasks in machine learning, pattern recognition, regression or time-series prediction. Even though architectures of ANNs evolve continuously, the building blocks, i.e. neurons, activation functions and the layered structure remain typical [17].

Neuron

A neuron is a node in a NN. Neurons receive inputs from incoming connections from other neurons and potentially themselves weighted by a factor w. Each neuron computes the sum of received and weighted signals and a bias term, passes it through an activation function and forwards the output through weighted connections to other neurons (see Figure 2.4).

Layer

Neurons in a NN are structured into layers. Typically, a NN consists of an input layer, one or more hidden layers and an output layer. Each layer receives its inputs

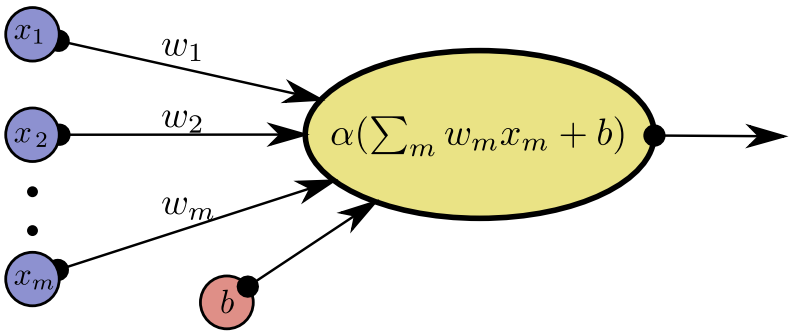

Figure 2.4: A basic neuron model.

from previous layers (and potentially itself) and passes its outputs to the subsequent layer. The input layer consists of D nodes (where D is the dimensionality of the input data) that each take a component of an input vector $\boldsymbol{x} \in \boldsymbol{X}$ [1], and forwards the processed output to the subsequent layer. Hidden layers are defined as intermediate layers situated between input and output layers, performing computations within the network. The number of neurons in a layer is referred to as *width*, the number of (hidden) layers as *depth*. A NN with more than three stacked hidden layers is called *deep neural network* (DNN). DNNs and the implications of using deeper architectures will be discussed in more detail in Section 2.3. For classification tasks, an output layer typically consists of one neuron per class. The computed value of an output neuron k represents the posterior probability of the input $\boldsymbol{x}$ belonging to class k. The computations of the incoming signals for a layer can be expressed in th form of a weight matrix $\boldsymbol{W}$ by concatenating the individual weights $\boldsymbol{w}$. The weight matrices $\boldsymbol{W}$ for all layers together with the bias vectors $\boldsymbol{b}$ constitute the parameters $\boldsymbol{\Theta}$ of the NN model [18].

Activation functions

An activation function computes a scaled output of the weighted sum of the inputs. In order to be able to discover non-linear classification boundaries, activation functions are commonly chosen to be also non-linear. In principal, any thinkable function could be used as an activation function, but in the course of time a few distinct functions have proven themselves of value (cf. Figure 2.5):

- *Sigmoid function*

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

  The sigmoid function, also referred to as *logistic function*, is a smoothened approximation of the step function used in early-day ANNs. Its output is

[1] Bold math symbols denote vectors and matrices in the following

bound in the range $[0, 1]$, making it suitable for the use in output neurons for classification tasks.

- *Hyperbolic tangent (tanh)*

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

The hyperbolic tangent shows a shape similar to the logistic function, vertically scaled for outputs in the range $[-1, 1]$.

- *Rectified linear unit (ReLU)*

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.3}$$

The ReLU enables sparse representations inside the NN by clipping negative values of x to 0. It also prevents the network from vanishing gradients and saturation problems that are common obstacles when working with deep neural networks [19].

- *Exponential linear unit (ELU)*

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \tag{2.4}$$

ELUs are different from ReLUs for values $x < 0$, setting the lower bound to $-\alpha$, where α is a tunable hyperparameter with the constraint $\alpha \geq 0$. ELUs push the mean activations closer to zero, which has been shown to speed up the learning process [20].

- *Softmax function*

$$\mathrm{softmax}(x)_j = \frac{e^{x_j}}{\sum_m e^{x_m}} \tag{2.5}$$

The softmax for a vector $\boldsymbol{x}$ is defined for each of its components x_j to satisfy the conditions $\mathrm{softmax}(x_j) > 0$ and $\sum_m \mathrm{softmax}(x_m) = 1$. It is commonly used for multiclass classification tasks, providing a normalized probability distribution over the output neurons [21].

Multilayer perceptron

With the previously introduced building blocks, it is now possible to build a simple neural network, e.g., the *multi layer perceptron* (MLP) [22]. In this simple NN architecture, the inputs are processed sequentially layer by layer, i.e., a layer only receives signals from its immediate predecessor (see Figure 2.6). A MLP takes a

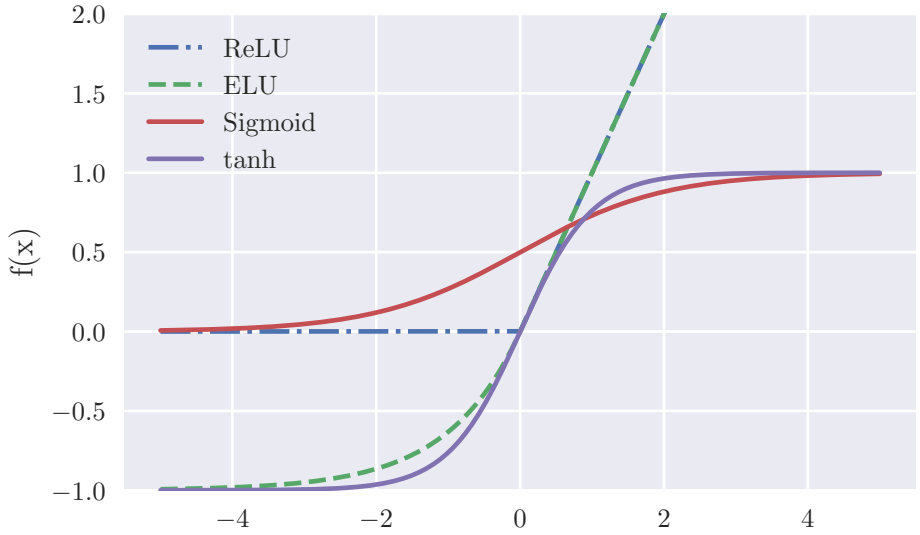

Figure 2.5: The sigmoid function, hyperbolic tangent (tanh), exponential linear unit (ELU, with $\alpha = 1.0$) and rectified linear unit (ReLU) activation functions. While the logistic and tanh functions saturate, both ELU and ReLU grow without upper bounds for positive values of x.

vector $\boldsymbol{x}$ as an input, calculates hidden activations $\boldsymbol{h}$ and outputs a vector $\hat{\boldsymbol{y}}$ as follows:

$$\boldsymbol{h} = \alpha_1(\boldsymbol{W}^{xh}\boldsymbol{x} + \boldsymbol{b}^h) \tag{2.6}$$

$$\hat{\boldsymbol{y}} = \alpha_2(\boldsymbol{W}^{h\hat{y}}\boldsymbol{h} + \boldsymbol{b}^{\hat{y}}) \tag{2.7}$$

where $\boldsymbol{W}^{ij}$ describes the weight matrix connecting two neighboring layers. Specifically, $\boldsymbol{W}^{xh}$ are the weights connecting input layer to hidden layer, $\boldsymbol{W}^{h\hat{y}}$ are the weights connecting hidden layer to output layer, $\boldsymbol{b}^l$ is a bias vector for layer l, and $\alpha_i(x)$ where $i \in \{1, 2\}$ are activation functions. Note that activation functions are applied element-wise when they are computed for all neurons in a layer simultaneously using matrix-vector notation as above. When multiple hidden layers are referred to, $\boldsymbol{h}^l$ is the subsequent layer to $\boldsymbol{h}^{l-1}$, where $l \in \{1, \ldots, L\}$ and L denotes the total number of hidden layers.

Weight initialization

At the beginning of the network construction, the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ are set to initial values. A common assumption is that positive and negative weights even out if they are properly initialized. However, the weights should not be universally set

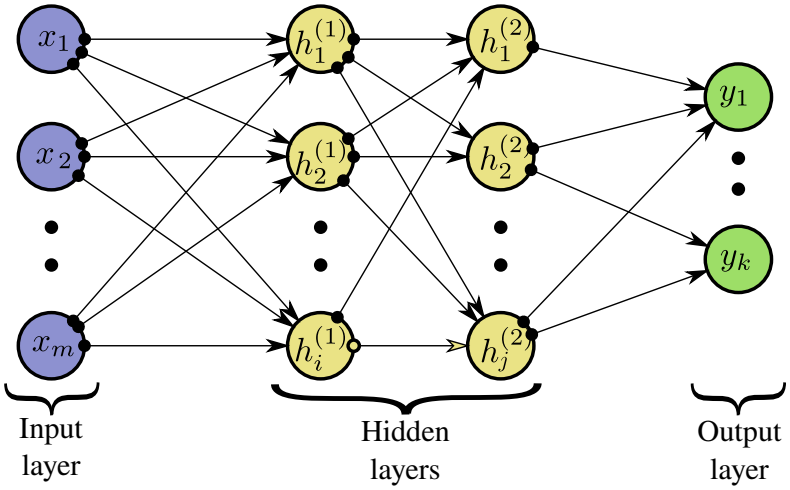

Figure 2.6: A multi layer perceptron with 2 hidden layers. Bias terms are omitted for clarity.

to zero because the backpropagated error and thus the gradient will be identical for all neurons in the network, since the output layer neurons show the same output. To break the symmetry between the weights, weights are usually initialized to small randomized values, keeping them close to zero [23]. The common procedure when using ReLU activation functions for the neurons is to draw a weight vector from a Gaussian with scaled variance relative to the number of input units n:

$$w_{ij}^{l} = \frac{\sqrt{n}}{2} \cdot \text{rand}(n) \tag{2.8}$$

where w_{ji}^{l} is the weight connecting the i^{th} neuron in layer l to the j^{th} neuron in layer $l+1$, rand(n) is a function sampling n values from a Gaussian distribution and the term $\frac{\sqrt{2}}{n}$ scales the variance, here with respect to the ReLU activation function.

Cost functions

For input $\boldsymbol{x}$ a prediction $\hat{\boldsymbol{y}}$ is computed at the output layer and evaluated using ground-truth $\boldsymbol{y}$ using a *cost function* $E(\boldsymbol{W}, \boldsymbol{b}; \boldsymbol{x}, \boldsymbol{y})$. The network is optimized (*trained*) by minimizing $E(\boldsymbol{W}, \boldsymbol{b}; \boldsymbol{x}, \boldsymbol{y})$ for all the training examples $(\boldsymbol{x}, \boldsymbol{y})$ in the training set:

$$E(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{n=1}^{N} E(\boldsymbol{W}, \boldsymbol{b}; \boldsymbol{x}_n, \boldsymbol{y}_n) \tag{2.9}$$

with N being the number of training examples. Common cost functions are:

- *Sum of squared errors (SSE)*

$$\mathrm{SSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{n=1}^{N} (\boldsymbol{y}_n - \hat{\boldsymbol{y}}_n)^2 \tag{2.10}$$

- *Root mean squared error (RMSE)*

$$\mathrm{RMSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sqrt{\frac{\sum_{n=1}^{N} (\boldsymbol{y}_n - \hat{\boldsymbol{y}}_n)^2}{N}} \tag{2.11}$$

- *Categorical cross entropy (CE)*

$$\mathrm{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} -\boldsymbol{y}_n^{(k)} \log(\hat{\boldsymbol{y}}_n^{(k)}) \tag{2.12}$$

  where K is the number of classes. For the case of binary CE with $K = 2$ follows:

$$\mathrm{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\frac{1}{N} \sum_{n=1}^{N} \boldsymbol{y}_n \cdot \log(\hat{\boldsymbol{y}}_n) + (1 - \boldsymbol{y}_n) \cdot \log(1 - \hat{\boldsymbol{y}}_n) \tag{2.13}$$

CE is generally used as the loss function for training NNs for classification tasks. CE has several properties that make it a suitable choice for the use as an error function when training a NN. First of all, the function is always non-negative, that is, $\mathrm{CE} > 0$ ($\forall\ \hat{\boldsymbol{y}}_n, \boldsymbol{y}_n \in [0, 1]$). The second advantage refers to the training process using the *backpropagation* algorithm with *gradient descent*, which will be discussed in the following paragraphs.

Gradient descent

The general idea of gradient descent is to follow the gradients on the surface of a cost function for a weight configuration of a NN while iteratively correcting the weights in the direction of the negative gradient slope, minimizing the cost function E. This is possible for NNs as they are constructed of differentiable components, which holds also true for the commonly used cost functions [18]. At the beginning of the network construction, the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ are initialized randomly. Then, the derivatives of the cost function E with respect to all of the weights and biases in the network are computed, i.e., $\frac{\partial E}{\partial w_{ji}^l}$ and $\frac{\partial E}{\partial b_{ji}^l}$, where w_{ji}^l is the weight connecting the i^{th} neuron in layer l to the j^{th} neuron in layer $l + 1$. The weights and biases can now be updated using, e.g., *stochastic gradient descent* towards the direction of the negative gradient:

$$\Delta w_i(\tau + 1) = -\alpha \frac{\partial E}{\partial w_i} \tag{2.14}$$

$$w_i(\tau + 1) = w_i(\tau) + \Delta w_i(\tau + 1), \tag{2.15}$$

where $\Delta w_i(\tau + 1)$ is the weight update, α is the *learning rate* that determines the update step size and τ is the iteration variable. The same update rule also applies to the biases b_i.

Backpropagation

To simplify the description of the previously mentioned backpropagation algorithm, the following notation is introduced: α' is the first derivative of activation function α, z_j^l is the weighted input to the j^{th} neuron in layer l and h_j^l is the activation of the j^{th} neuron in layer l, i.e., $h_j^l = \alpha(z_j^l)$ and

$$z_j^l = \sum_i w_{ji}^l h_i^{l-1} + b_j^l. \tag{2.16}$$

Backpropagation is a method used in ANNs to compute the error contributed by every individual neuron in the network [18]. It is also called *backpropagation of errors*, because the error is computed at the output layer neurons and propagated back through the network.
Formally, the error δ_j^L for each neuron j in the output layer (the output layer being layer L) is computed as:

$$\delta_j^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial h_j^L}\frac{\partial h_j^L}{\partial z_j^L} = \frac{\partial E}{\partial h_j^L}\alpha'(z_j^L) \tag{2.17}$$

Based on the errors δ_j^{l+1}, the backpropagated errors δ_j^l in the previous layer l can be computed:

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_i \frac{\partial E}{\partial z_i^{l+1}}\frac{\partial z_i^{l+1}}{\partial z_j^l} = \sum_i w_{ij}^{l+1}\delta_i^{l+1}\alpha'(z_j^l), \tag{2.18}$$

where the following relationships were used:

$$\frac{\partial z_j^{l+1}}{\partial z_j^l} = \frac{\partial}{\partial z_j^l}\sum_i w_{ij}^{l+1}\alpha(z_j^l) + b_i^{l+1} = \sum_i w_{ij}^{l+1}\alpha'(z_j^l), \tag{2.19}$$

$$\delta_i^{l+1} = \frac{\partial E}{\partial z_i^{l+1}}. \tag{2.20}$$

Now, by using the expression

$$\frac{\partial z_j^l}{\partial w_{ji}^l} = \frac{\partial}{\partial w_{ji}^l}\sum_i w_{ji}^l h_i^{l-1} + b_j^l = h_i^{l-1} \tag{2.21}$$

and the definition of δ_j^l, the gradients $\frac{\partial E}{\partial w_{ji}^l}$ as a function of the error δ_j^l can be expressed:

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{ji}^l} = h_i^{l-1}\delta_j^l \tag{2.22}$$

Analogously, the gradients $\frac{\partial E}{\partial b_j^l}$ of the cost function with respect to the bias terms are:

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l. \tag{2.23}$$

Usually, variations of stochastic gradient descent (SGD) such as batch gradient descent or mini-batch gradient descent are used, handling multiple input-output pairs $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ at once. This effectively increases computation speed by exploiting the fast matrix multiplication implementations available on graphics processing units (GPUs) [24]. Besides the mentioned gradient descent variants, there exist further optimization algorithms such as RMSprop [25], ADADELTA [26] and Adam [27]. The latter is used in the implementations of this work.

Adam The optimization algorithm Adam (derived from adaptive moment estimation) was developed by Kingma et al. [27] and is an extension to stochastic gradient descent (SGD) that is widely used for optimizing deep learning parameters in computer vision and natural language processing. While SGD maintains a single learning rate α for all weight updates without adaptation during training, Adam employs an individual learning rate for each weight and separately adapts in training time from estimates of first and second moments of the gradients. The algorithm is described in full detail by its authors in [27].

2.3 Deep Learning

Deep learning has achieved breakthroughs in a large range of real-world applications in various domains thanks to the progress made in recent years. The success of deep neural networks can be attributed to both technological and scientific advances. On the one hand, training these networks with possibly millions of parameters in a timely manner is made possible by using graphics processing units (GPUs) instead of central processing units (CPUs). Fast and efficient matrix and vector multiplications (common in neural networks) are the strengths of today's powerful GPUs when compared to CPUs [24]. On the other hand, there have also been theoretical insights on the advantage of deep networks over shallow networks, considering that both have an identical overall number of neurons in the hidden layers. The reason for that is the exponential growth of the expressiveness of models with increasing depth [28, 29]. Furthermore, rapid prototyping and implementation of new ideas and architectures is enabled by novel software frameworks, such as Google's TensorFlow [30], Theano [31], Microsoft's CNTK [32], Facebook's PyTorch [33], Keras [34] or Caffe [35].
In contrast to deep learning, classic machine learning approaches for classification tasks require the design of hand-crafted discriminative features which are fed to a statistical classifier, such as linear discriminative analysis (LDA) [36] or support

vector machine (SVM) [37]. The design of hand-crafted features, however, requires in-depth knowledge of the problem domain and finding highly expressive features in complex tasks such as computer vision or natural language processing is challenging. In addition to that, with a growing number of features the dimensionality of the problem increases, which leads to the *curse of dimensionality*[2].
With deep learning the way to approach complex problems has changed. The multiple hidden layers allow the automatic discovery of complex features by hierarchically learning abstract representations in the first few layers and more specific characteristics in the following layers. The number of distinguishable features in deep architectures grows exponentially with the number of model parameters, while showing good generalization capabilities, providing that sufficient labeled data is available [29].
In the following, common deep learning models are explained, including feedforward neural networks, convolutional neural networks and long short-term memory, which are used in this thesis.

2.3.1 Feedforward Neural Networks

A feedforward neural network (FNN) is composed of fully-connected layers as previously described in section 2.2 for the most simple FNN, i.e., the multilayer perceptron (MLP). Albeit its relatively simple structure, a MLP with only one hidden layer can produce the characteristics of any smooth mathematical function with high precision, given that a sufficient amount of non-linear neurons are used in the hidden layer, which is stated by the *universal approximation theorem* [39]. The problem is that the required number of hidden layer neurons is a priori unknown. However, MLPs with multiple hidden layers have shown satisfying performance in handwritten digit recognition [22] and speech recognition [40], to name a few examples.
A different type of FNN architecture is described in the following section: The convolutional neural network.

2.3.2 Convolutional Neural Networks

The main component of a convolutional neural network (CNN) is the convolutional layer. The idea behind convolutional layers assumes that a locally learned feature for a given input (typically two-dimensional, e.g., images) should also be useful in other regions of that same input, e.g., an edge detector that was found useful in some part of an input image should also prove useful in other parts of the image as a general feature extraction stage. The learning of features such as differently oriented edges, curves or color blobs is attained by sliding, or more precisely, convolving, a set of filters over the input dimensions, which for image-like inputs would

[2]Having only a limited amount of training data, the predictive capabilities of the model are reduced with increasing dimensionality, which is also known as *Hughes phenomenon* [38]. This often results in poor generalization capabilities of the model for complex tasks.

be width and height for example (see figure 2.7 for hierarchically learned filters in subsequent layers in a facial recognition task). The filters cover the full depth of

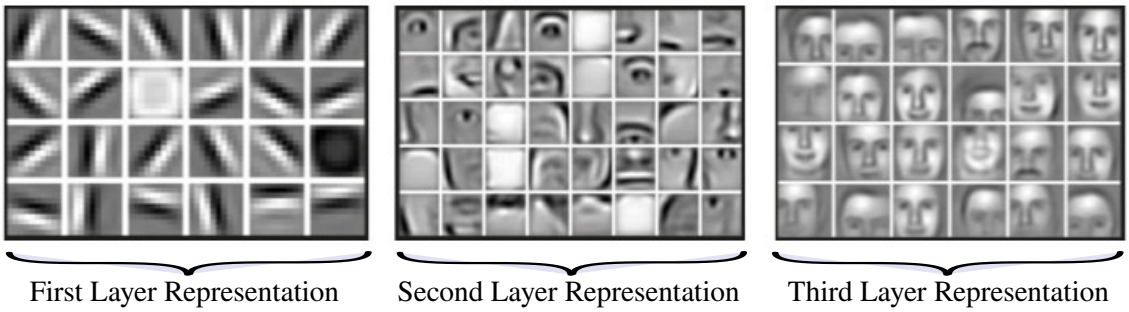

Figure 2.7: Hierarchical representation of learned features in a CNN used for facial recognition tasks. Illustration adapted from [41].

the input data, meaning that filters for colored images in RGB format need to cover all three color channels in the depth dimension. The output of a single filter is a two-dimensional activation at every covered spatial location.
A convolutional forward pass can be expressed as a single matrix-vector multiplication for optimization purposes:

$$\boldsymbol{y} = \boldsymbol{W} \cdot \boldsymbol{x} \tag{2.24}$$

where the n-dimensional input matrix $\boldsymbol{x}$ and the convolutional filters are flattened into vectors. The filters are arranged in column-wise order to form weight matrix $\boldsymbol{W}$. Tunable hyperparameters of CNNs include the number of filter kernels K, their size F (e.g., filter dimensions are 2×2 for $F = 2$), the stride indicating the spatial shift while sliding the filter, and how to handle the regions at the border, i.e., type and size of padding.
Typical CNN architectures are composed of consecutive convolutional layers, ReLU or ELU activations, batch normalization layers, pooling layers, dropout layers, fully-connected layers and a softmax activation in the output layer. A simple CNN without dropout and batch normalization is illustrated in figure 2.8. Pooling layers are described in the paragraph below. CNNs have achieved outstanding results, surpassing human-level classification accuracies in the field of computer vision, e.g., on the 1000-class ImageNet competition [6].

Pooling layer

Pooling layers are commonly used in CNNs to down-sample the output of convolutional layers by using a sliding filter (typicall mean or max-filter). For example, a filter of size 2×2 and stride of 2 sub-samples an input by a factor of 2 in both height and width, which is often applied in CNN architectures to combat overfitting and improve translational invariance. Pooling layers have two hyperparameters, namely filter size F and stride S.

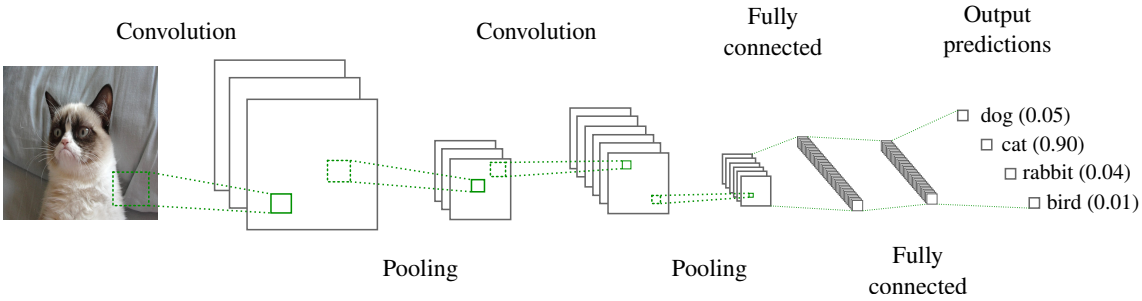

Figure 2.8: An example of a convolutional neural network used for image classification tasks.

A FNN-based architecture such as a CNN, however, exhibits a few drawbacks when processing sequential input data like audio, video or natural language, because the input samples are processed independently from one another. Recurrent neural networks (RNNs) which are described in the following section, are a more suitable choice for sequential input data.

2.3.3 Vanilla Recurrent Neural Networks

For some applications such as natural language processing (NLP) or speech recognition, important information is provided in the context of the input data due to its sequential nature. As traditional FNN-based architectures are not well-suited for handling sequential data, recurrent neural networks (RNNs) present a solution by introducing feedback connections within layers. Over the past years, several types of RNNs have been developed, such as Elman RNNs [42], long short-term memory (LSTM) [43] and gated recurrent unit (GRU) [44] networks.

In general, for a sequence of input vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ a RNN computes a sequence of hidden activations $\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T\}$ and output vectors $\{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_T\}$ for time steps $t \in [1, \ldots, T]$ as following:

$$\boldsymbol{h}_t = \alpha_1(\boldsymbol{W}^{xh}\boldsymbol{x}_t + \boldsymbol{W}^{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}^h) \tag{2.25}$$

$$\hat{\boldsymbol{y}}_t = \alpha_2(\boldsymbol{W}^{h\hat{y}}\boldsymbol{h}_t + \boldsymbol{b}^{\hat{y}}) \tag{2.26}$$

where $\boldsymbol{W}^{ij}$ denotes the weight matrices connecting layers i and j, b^j are bias terms and α_i (with $i \in \{1, 2\}$) activation functions.
In a RNN with multiple stacked hidden layers, the hidden layers receive the activation of the previous hidden layers per timestep (see Figure 2.9). The addition of feedback connections on layers enables a RNN to let information flow through the time steps, whereby the hidden layers produce activations that act as *memory* in the network. The hidden layers step by step build up an *internal state* through their activation vectors $\boldsymbol{h}_t$. Hence, the output $\hat{\boldsymbol{y}}_t$ at time step t becomes a function of all

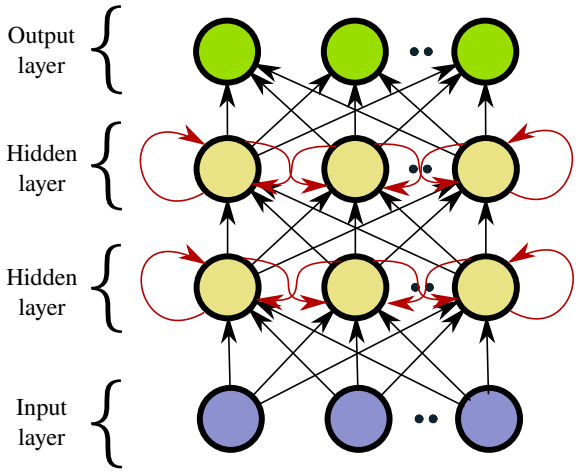

Figure 2.9: Basic architecture of a recurrent neural network with two hidden layers. Feedback connections are highlighted in red. Bias terms are omitted for clarity.

received inputs $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$ and hidden activations $\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_t\}$ before the timestep t. Based on those characteristics, RNNs are especially well-designed for processing sequential data. The simple RNN described here is known as *vanilla RNN*.

Backpropagation through time

Backpropagation through time (BPTT) is an extension to the ordinary backpropagation algorithm (see Section 2.2 for details). The idea is to unroll a RNN in time and then use backpropagation, treating the RNN as if it were a FNN. Figure 2.10 illustrates the equivalence of a RNN and its unrolled representation, which is analogous to a FNN having a layer for each time step while sharing weights [45].

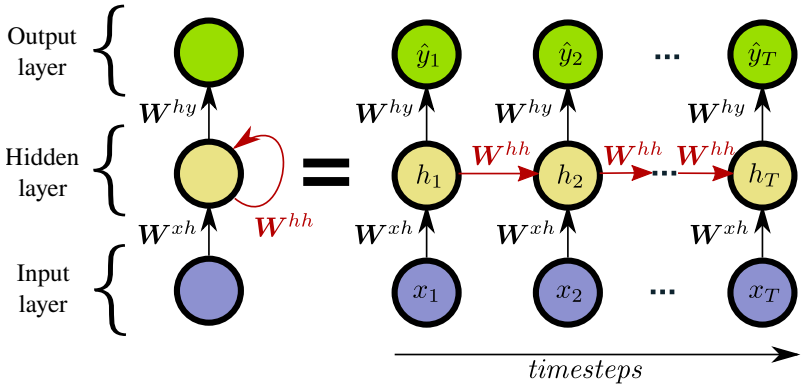

Figure 2.10: Left-hand side: A recurrent neural network with a single hidden layer with one single neuron. Right-hand side: The same network unrolled in time over T timesteps.

Vanishing and Exploding Gradients

One major difficulty with RNNs is the vanishing and exploding gradient problem that can occur when training vanilla RNNs, which rendered RNNs unusable for many use cases in its early years [46]. The reason for that problem is explained in the following paragraph.

The cost function E_t is computed up to timestep t based on outputs $\hat{\boldsymbol{y}}_t$ and labels $\boldsymbol{y}_t$ for a sequence where t ranges from 1 to T and $E = \sum_{t=1}^{T} E_t$ is defined as the cost function for the whole sequence. In contrast to backpropagation in FNNs, the gradients for cost E_t with respect to the weights $\boldsymbol{W}^{hh}$ need to be propagated multiple times through the same weight matrix $\boldsymbol{W}^{hh}$, because output $\hat{\boldsymbol{y}}_t$ depends on hidden state activations $\boldsymbol{h}_t$, which themselves depend on all previous hidden state activations $\boldsymbol{h}_1$, $\boldsymbol{h}_2$, ..., $\boldsymbol{h}_t$. The gradients thereby become smaller and smaller (vanishing gradients) or larger and larger (exploding gradients) according to the chain rule in backpropagation, which makes efficient training unfeasible for a larger number of timesteps. In general, vanishing or exploding gradients can occur in any type of deep neural network, but RNNs are particularly prone to that problem due to unrolling the network for BPTT, which effectively results in particularly deep NNs.
While exploding gradients can be attenuated by clipping gradients whose norms surpass a certain threshold [47], vanishing gradients are a more challenging problem to solve. However, several methods have been developed to overcome the limitations of vanilla RNN, e.g., novel neural network architectures such as the long short-term memory, which will be explained in the following section.

2.3.4 Long Short-Term Memory Networks

To address the vanishing gradient problem in vanilla RNN, Hochreiter et al. proposed the long short-term memory (LSTM) network in 1997 [43] by replacing the simple neurons with LSTM units (also called *memory cells*).

LSTM memory cell

A memory cell consists of four main components: an *input gate*, a *neuron with a self-recurrent connection*, a *forget gate* and an *output gate* (see the illustration of a LSTM cell in Figure 2.11). The self-recurrent connection with a weight of 1.0 bars outside interference and thus makes sure that the state of a memory cell may remain constant from one timestep to another (represented by the top straight line denoted $\boldsymbol{C}_t$ in Figure 2.11). The gates regulate addition and removal of information to/from the memory cell. They are composed of layers using either sigmoid or tanh activations. The input gate decides whether the state of the memory cell should be altered by the input signal or not. The output gate on the other hand decides if the memory cell state should have an effect on the output (and therefore on following

LSTM neurons). Eventually, the forget gate regulates the memory cell's ability to remember or forget its previous cell state through its self-recurrent connection [48].

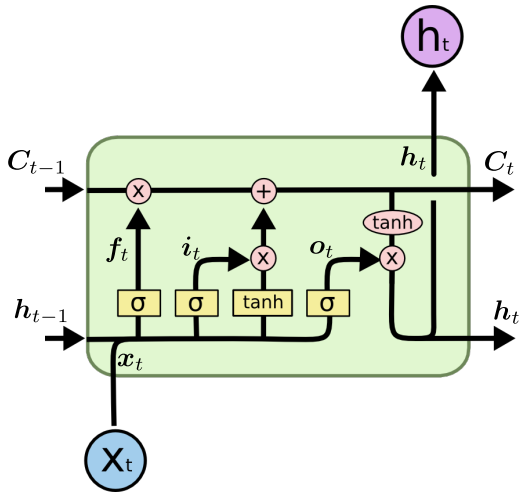

Figure 2.11: Long short-term memory cell. Illustration adapted from [48].

The hidden activation $\boldsymbol{h}_t$ in a LSTM layer is computed by the following set of equations:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^{xi}\boldsymbol{x}_t + \boldsymbol{W}^{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}^i) \tag{2.27}$$
$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^{xf}\boldsymbol{x}_t + \boldsymbol{W}^{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}^f) \tag{2.28}$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tanh(\boldsymbol{W}^{xc}\boldsymbol{x}_t + \boldsymbol{W}^{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}^c) \tag{2.29}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^{xo}\boldsymbol{x}_t + \boldsymbol{W}^{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}^o) \tag{2.30}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t) \tag{2.31}$$

where $\boldsymbol{i}_t$ denotes the input gate, $\boldsymbol{f}_t$ the forget gate, $\boldsymbol{c}_t$ the cell state, $\boldsymbol{o}_t$ the output gate, $\sigma(.)$ the sigmoid activation function, $\boldsymbol{W}^{ij}$ the weight matrices and $\boldsymbol{b}^j$ the bias terms. The operator " $\circ$ " denotes element-wise multiplication.

The forget gate was added to the original LSTM architecture in 1999 by Gers et al. [49]. This extended LSTM version is widely used today, which is one of the reasons it was also chosen to be used in this thesis.
The vanishing gradient problem is mitigated by the self-recurrent connection $\boldsymbol{C}_t$ of the cell with a weight of 1.0, which keeps the gradients from being scaled multiple times and thus avoids previously described problems of vanilla RNNs (see Section 2.3.3).

2.3.5 Tuning Deep Neural Networks

There are several techniques that can help to improve classification accuracy or generalization capabilities of a deep learning model. In the following paragraphs a few commonly used methods will be briefly described.

Batch normalization Batch normalization is typically applied to normalize intermediate representations between layers, which has been shown to improve generalization and accuracy, especially for CNNs [50].

Dropout Dropout layers combat overfitting by randomly disabling a certain percentage of neurons in a layer. This effectively makes sure that a network learns generalized features rather than relying on individual neural connections. Dropout is only used during the training phase and turned off for validation and test [51].

Regularization Regulizers pursue the same goal as dropout (reduce overfitting) by penalizing weights. Specifically, L2 regularization introduces a term $\lambda \sum_i w_i^2$ to the cost function, where λ is a factor controlling the regularization strengths, which effectively punishes the overuse of weights in a network [52].

Data augmentation As the performance of deep learning methods heavily relies on large amounts of data, using more data likely improves the deep learning model. Data augmentation aims to artificially produce more training data from available data. For the case of image data, it is for example possible to rotate, scale or flip the images without changing the meaning. By feeding augmented data to the network, the network learns some degree of invariance to this type of image transformations [6].

Cross validation When training a machine learning model, a method to choose between different models is needed. A common metric for the model's performance is to measure its accuracy on unseen data. For small amounts of data, however, there can be a rather big variability in the validation set. Therefore, *k-fold cross-validation* is commonly used. For $k = 10$ a dataset is split into 10 equally sized subsets. One of the subsets is used for validation, while the remaining 9 subsets are used for training, which is done iteratively until every subset was once used for validation. The validation accuracies are then averaged, resulting in a representative measure of the models performance.
For deep learning, however, it is normally avoided to use cross-validation because of the cost associated with training k different models on large amounts of data. Instead of using cross-validation, it is common practice to choose random subsets of the data for training, validation and testing (usually 70% / 15% / 15%). The model is trained until the minimum of the validation loss (see also *early stopping*,

explained in [52]) is found and then tested on the test data split, measuring the generalization capabilities of the model.

2.4 Related Work

This section presents an overview of related work in the field of BCI research. First, the state of the art of EEG decoding methods is described. Then, novel EEG decoding approaches based on deep learning are reviewed.

Traditional Machine Learning Approaches

During the past two decades, significant improvement in performance of motor imagery-based BCIs has been achieved and various approaches for decoding EEG signals have been investigated. Most of the decoding methods are based on traditional machine learning algorithms, such as support vector machines (SVM) [2, 53] and linear discriminant analysis (LDA) [54, 53, 55]. For instance, Yong et al. presented in their work the classification of three classes of motor imagery EEG data within the same limb [2]. The three motor imagery tasks involved imaginary grasp and elbow movements, and a rest state. Best results were achieved using SVM on relative event-related desynchronziation/resynchronization (ERD/ERS) features (cf. Section 2.1.1). An accuracy of 66.9% was achieved for binary classification (grasp vs. elbow movements). The accuracy dropped to 60.7% when using the rest state as the third class.

Yang et al. implemented an filter bank common spatial pattern (FBCSP) approach that generates EEG features in various frequency ranges [56]. They tested the FBCSP approach on BCI competition IIa dataset with 9 subjects on four motor imagery classes (left-, right- and both-hand movements and rest). They achieved a cross-validated accuracy of 67.01% ($\pm$16.20%), which is the state of the art for four-class motor imagery classification using statistical machine learning methods.

Deep Learning Approaches

Stagnating decoding accuracies in both offline and online setups have led the way to novel approaches, such as deep learning. Considering that neural activity measured by EEG can be regarded as highly dynamic, non-linear time series data, recurrent neural networks and particularly LSTM appear to be the tool of choice for modeling the temporal characteristics of brain activity. While LSTM-based models have reached state-of-the-art performance in sequential data processing tasks such as natural language processing, CNNs excel in computer vision and speech recognition tasks. This raises the question whether those architectures are suitable for EEG decoding purposes.

Lawhern et al. developed and investigated the use of a generic CNN model called *EEGNet* that classifies EEG signals for different BCI tasks, such as P300 visual-evoked potentials (P300), error-related negativity responses (ERN), movement-related cortical potentials (MRCP), and sensory motor rhythms (SMR, see Section 2.1.1 for details) [8]. Although the CNN architecture was not designed to excel on a single task, EEGNet performs comparable to FBCSP. However, the results vary largely across different subjects and the authors do not provide detailed numeric results, but only show the relative performance. Their main finding is that a simple CNN architecture can provide robust performance across different BCI tasks.

Stober et al. introduced and compared methods for learning expressive features with deep learning techniques [57]. They addressed the problem of limited EEG data availability by training convolutional auto-encoders on cross-trial encoded EEG data, which helped to find stable features that are present in all trials. The dataset used for measuring their model's performance was the OpenMIIR dataset, which consists of EEG recordings from participants listening to music.

Alex Greaves implemented a RNN and a fully-connected NN to classify EEG that was recorded when participants where viewing either 2D or 3D stimuli [58]. Even though RNNs are designed to process sequential data (such as EEG), the implemented RNN model showed unsatisfying performance and was outperformed by a regular FNN that reached an accuracy of 72% on the binary classification task.

Bashivan et al. developed a recurrent-convolutional neural network inspired by video classification techniques [59]. They transformed EEG time-series data to a sequence of electrode-location preserving EEG images. The model was evaluated on a mental load classification task that elicited event-related potentials (ERP) in the EEG recordings. A test error of 8.89% was reached on the four-class problem. However, as ERP tasks are relatively easy to classify compared to motor imagery tasks, the significance of the results for an active BCI is disputable. Furthermore, the transformation of EEG channels to multiple topology-preserving EEG images is computationally expensive and therefore not suitable for real-time applications.

Schirrmeister et al. investigated the design choices of CNN architectures for decoding motor imagery movements from raw EEG data [60]. Two models were presented in their work, that is, a deep and a shallow CNN model. By including recent advances from the field of deep learning, such as batch normalization and exponential linear unit activations, the authors obtained accuracies of 71.90% and 70.10% on BCI dataset IVa for deep and shallow CNN models, respectively. As this work was published towards the end of this thesis project, it did not influence the design choices for proposed CNN and LSTM models described in later sections of this thesis. However, both the deep and shallow CNN models have been implemented in this work for benchmarking purposes. Their architectures are explained in the following paragraphs.

Deep Convolutional Neural Network The deep CNN model developed by Schirrmeister et al. [60] was inspired by successful ImageNet competition contenders, such as AlexNet [6]. The model consists of four convolutional-max-pooling blocks, whereas the first block is specifically designed for handling EEG data. The output layer is a standard dense softmax classification layer. The first convolutional block was split in two because EEG is often recorded using many electrodes (i.e., channels), compared to only three channels in RGB images. The filters of the first layer learn temporal convolution kernels, while filters in the second layer operate spatially in 2D across the learned temporal filters. Because no activation function is applied between the first two layers, they could in general be combined into a single layer. However, splitting the operations between two layers enhanced the model's performance [60]. For the subsequent convolutional blocks the ELU activation function was used. Table 2.1 shows the implementation of the deep CNN model for three EEG channels as it was implemented in this thesis for benchmarking purposes.

Shallow Convolutional Neural Network The shallow CNN architecture developed by Schirrmeister et al. [60] was inspired by the FBCSP decoding pipeline, which is especially useful for decoding band power features. The first two layers of the shallow CNN model perform temporal and spatial convolutions as in the deep CNN model, which is similar to the subsequent bandpass and spatial filter stages in FBCSP. However, the filter kernels of the temporal convolution layer are larger than in the deep CNN model (25×1 vs. 10×1). Following the first two layers, a squaring activation function, a mean pooling layer and a logarithmic activation function are applied. Those operations are analogous to trial log-variance computations in FBCSP (see the original work for details [60]). Table 2.2 shows the implementation of the shallow CNN model for three EEG channels as it was implemented in this thesis for benchmarking purposes.

Layer	Input	Operation	Output	Parameters
1	$E \times T$	$25 \times$ Conv2D (1×10)	$E \times 1015 \times 25$	275
2	$E \times 1015 \times 25$	Reshape	$E \times 1015 \times 25 \times 1$	-
	$E \times 1015 \times 25 \times 1$	$25 \times$ Conv3d $(3 \times 1 \times 25)$	$1 \times 1015 \times 1 \times 25$	1,900
	$1 \times 1015 \times 1 \times 25$	BatchNorm	$1 \times 1015 \times 1 \times 25$	100
	$1 \times 1015 \times 1 \times 25$	ELU	$1 \times 1015 \times 1 \times 25$	-
	$1 \times 1015 \times 1 \times 25$	MaxPool2D (3×1)	$338 \times 25 \times 1$	-
	$338 \times 25 \times 1$	Dropout (0.5)	$338 \times 25 \times 1$	-
3	$338 \times 25 \times 1$	$50 \times$ Conv2D (10×25)	$329 \times 1 \times 50$	12,550
	$329 \times 1 \times 50$	BatchNorm	$329 \times 1 \times 50$	200
	$329 \times 1 \times 50$	ELU	$329 \times 1 \times 50$	-
	$329 \times 1 \times 50$	MaxPool2D (3×1)	$109 \times 1 \times 50$	-
	$109 \times 1 \times 50$	Dropout (0.5)	$109 \times 1 \times 50$	-
4	$109 \times 1 \times 50$	Reshape	$109 \times 50 \times 1$	-
	$109 \times 50 \times 1$	$100 \times$ Conv2D (10×50)	$100 \times 1 \times 100$	50,100
	$100 \times 1 \times 100$	BatchNorm	$100 \times 1 \times 100$	400
	$100 \times 1 \times 100$	ELU	$100 \times 1 \times 100$	-
	$100 \times 1 \times 100$	MaxPool2D	$33 \times 1 \times 100$	-
	$33 \times 1 \times 100$	Dropout (0.5)	$33 \times 1 \times 100$	-
5	$33 \times 1 \times 100$	Reshape	$33 \times 100 \times 1$	-
	$33 \times 100 \times 1$	$200 \times$ Conv2D (10×100)	$24 \times 1 \times 200$	200,200
	$24 \times 1 \times 200$	BatchNorm	$24 \times 1 \times 200$	800
	$24 \times 1 \times 200$	ELU	$24 \times 1 \times 200$	-
	$24 \times 1 \times 200$	MaxPool2D (3×1)	$8 \times 1 \times 200$	-
6	$8 \times 1 \times 200$	Flatten	1600	-
	1600	Softmax	K	3,202
Total				$268,977$

Table 2.1: Deep CNN architecture as proposed by Schirrmeister et al. [60] (re-implemented in this work), where E is the number of channels, T is the number of timesteps and K is the number of classes. Input and Output sizes are shown for cropped training with $E = 3$ (electrodes C3, C4 and Cz) and $T = 1024$ for window size of 4 seconds; binary classification with two classes for $K = 2$.

Layer	Input	Operation	Output	Parameters
1	$E \times T$	$40 \times$ Conv2D (1×25)	$E \times 1000 \times 40$	1,040
	$E \times 1000 \times 40$	Dropout (0.5)	$E \times 1000 \times 40$	-
2	$E \times 1000 \times 40$	Reshape	$E \times 1000 \times 40 \times 1$	-
	$E \times 1000 \times 40 \times 1$	$40 \times$ Conv3D $(3 \times 1 \times 40)$	$1 \times 1000 \times 1 \times 40$	4,840
	$1 \times 1000 \times 1 \times 40$	BatchNorm	$1 \times 1000 \times 1 \times 40$	160
	$1 \times 1000 \times 1 \times 40$	x^2-Activation	$1 \times 1000 \times 1 \times 40$	-
	$1 \times 1000 \times 1 \times 40$	Dropout (0.5)	$1 \times 1000 \times 1 \times 40$	-
3	$1 \times 1000 \times 1 \times 40$	Reshape	$1000 \times 40 \times 1$	-
	$1000 \times 40 \times 1$	AvgPool2d (75×1)	$62 \times 40 \times 1$	-
	$62 \times 40 \times 1$	$\log(x)$-Activation	$62 \times 40 \times 1$	-
4	$62 \times 40 \times 1$	Flatten	2480	-
	6144	Softmax	K	4,962
Total				$10,922$

Table 2.2: Shallow CNN architecture by Schirrmeister et al. [60] (re-implemented in this work), where E is the number of channels, T is the number of timesteps and K is the number of classes. Input and Output sizes are shown for cropped training with $E = 3$ (electrodes C3, C4 and Cz) and $T = 1024$ for window size of 4 seconds; binary classification with two classes for $K = 2$.

Chapter 3

Methods

This chapter first explains the general experimental design for brain-signal decoding using deep learning and the datasets used in this work. The second part explains proposed LSTM- and CNN-based architectures and their implementations. Finally, performance of the models is evaluated in the last section.

3.1 Experimental Design

This work aims to further investigate the potential of deep learning methods to classify binary motor imagery movements in EEG signals. Investigated approaches include the long short-term memory (LSTM) recurrent networks as well as convolutional neural networks (CNNs) described in Sections 2.3.4 and 2.3.2, respectively. In the following, first, the data used in this study is described. Then, in the subsequent section design choices and implementations for both LSTM and CNN networks are explained.

3.1.1 Data

Since deep learning models often consist of hundreds of thousands of tunable parameters, a huge amount of data is needed to be able to accurately optimize the models and achieve good performance. For this reason, and since none of the publicly available EEG datasets offer enough data samples for multiple subjects, we decided to record our own data. This dataset is referred to as *NST data* afterwards. As NST data is not yet available to the public, the Graz data set B [13] is additionally used to validate our results and rule out any possible bias. The Graz dataset B recording paradigm is explained in Section 2.1.2.

NST Recording paradigm

This data set was acquired over the course of the duration of this thesis at the *Assistant Professorship for Neuroscientific System Theory* (NST) at *Technical University*

of Munich following a recording protocol by Zied Tayeb. The data set consists of EEG data from 20 healthy subjects with normal or corrected-to-normal vision. The participants were PhD students, master students, student research assistants or volunteers. During the experiment the participants were sitting in an armchair located 1 meter away from the flat screen they needed to watch. Each subject underwent four recording sessions, whereby the first three contain training data without feedback, and the fourth contains data with random on-screen feedback. Each session consists of 72 trials, 24 for each class (left hand, right hand and both hand movements), which makes 216 trials per subject without feedback in total. The participants were instructed to imagine familiar movements that they could imagine best, such as squeezing a ball, lifting a water bottle, or passively moving their arm sideways. Each trial has a duration of 10 seconds and after each session the participants were asked to take a 10 minute break. At the beginning of a trial ($t = 0$s) the screen is black. At $t = 3$s a green fixation dot is displayed to alert the subject. At $t = 4$s a visual cue in form of a red arrow pointing either to the right or the left-hand side or both is displayed on the screen. At $t = 4.5$ s a short acoustic signal (1 KHz, 500 ms) is played, signaling to the subject to start performing motor imagery and closing his/her eyes. At $t = 8.5$s the acoustic signal is played again, instructing the subject to open his/her eyes and relax. Up until $t = 10$s there is a pause after which the trial has finished (see Figure 3.1 for illustration). No movement execution is requested and the subjects are instructed to keep their hands fully supported on the arm chair in resting position. The type of cue (left, right, or both) is selected randomly. However, at the end of a session the same number of trials was recorded for each class. Figure 3.2 shows the recording setup at NST for better understanding. The

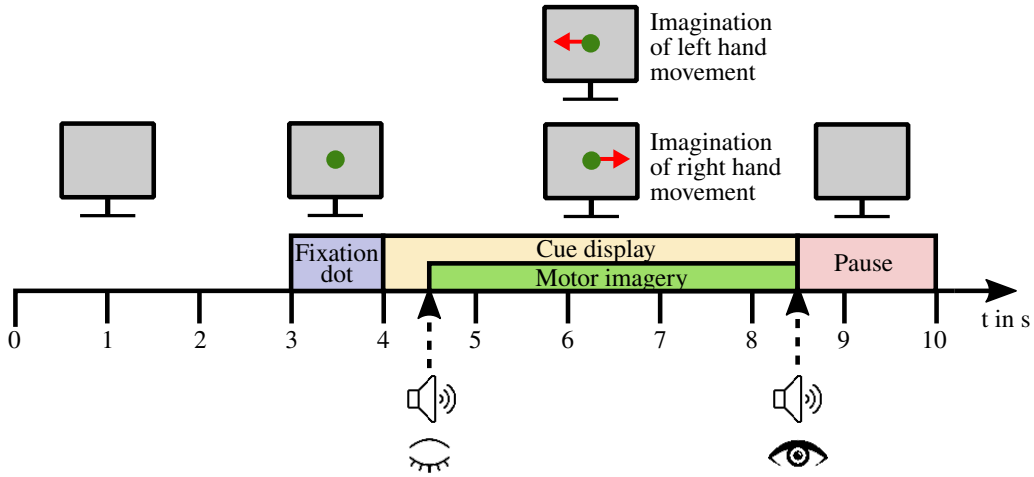

Figure 3.1: Illustration of the NST recording paradigm for motor imagery EEG data acquisition.

recordings were performed using a *g.USBamp* amplifier (*g.tec medical engineering GmbH, Austria*) on electrodes *C3, C4* and *Cz*. The reference electrode was placed

on the right earlobe and ground was on electrode *AFz*. The sampling frequency was set to 256 Hz; the power line frequency in Germany is at 50 Hz. Recorded data was saved as is without any filtering.
Note that in this thesis only the first three sessions without feedback are used.

Figure 3.2: EEG recording setup at NST.

3.2 Implementation

This section explains the implementation of proposed models for decoding imagery movements from EEG data. The characteristics of this classification problem support the assumption that deep neural networks are an exceptionally promising approach for decoding EEG data. First, it is challenging to design discriminative features from scratch because there is a high variability between samples. Second, even though the potential of collecting large amounts of data is limited, there are efficient methods for data augmentation (see Section 3.2.2). This work approaches the classification problem with two different types of deep neural networks: LSTM networks and CNN. While the latter has been very recently evaluated on motor imagery data [60] and to a certain degree on other types of EEG signals (event-related potential, visually-evoked potentials etc.), the former has not yet been evaluated for its use in classifying EEG signals.

In the first part, preprocessing and augmentation of the data are described, which is identical for all proposed deep learning approaches. In the subsequent parts, first the proposed LSTM and then the proposed CNN model are explained.

3.2.1 Signal Preprocessing

Raw EEG signals naturally have a low signal-to-noise ratio and suffer from interference from various noise sources, such as power line noise, electromyography (EMG)

and electrooculography (EOG) artifacts from surrounding muscle and eye movements, and also bad electrode placement on the scalp. It is therefore a common practice to filter EEG data prior to any further feature extraction or classification steps. The following preprocessing stages were implemented in Python using `NumPy` and `SciPy.Signal` libraries.

1. **Notch filter**
   In order to eliminate power line noise, a notch (or band stop) filter was applied in the frequency range of 50 Hz with a quality factor of $Q = 30$.

2. **High pass filter**
   Over the course of the recording session the impedance between electrodes and scalp slowly changes due to minor movements of skin against electrodes and gel. This results in a slow baseline drift in the EEG data, which can be removed by either using statistical detrending methods or by simply applying a high pass filter to the data. Here, the data was high pass filtered with a cutoff frequency of 0.5 Hz to remove baseline drift.

3. **Band pass filter**
   Signal characteristics that are typical for motor imagery are usually found in the frequency ranges of α- and β-rhythms ($8 - 13$ Hz and $13 - 30$ Hz, respectively). Interfering EMG and EOG signals are usually found in higher frequency ranges. Therefore, a band pass filter is applied to the EEG data. However, for deep learning methods it is desired to perform a minimum of signal preprocessing because unknown features in other frequency bands might be found. In this work a passband from 2 to 60 Hz in a digital zero-phase Butterworth filter of order 5 is used.

4. **Six-sigma clipping**
   Unintentional movement during a recording session can lead to high voltage spikes in the recorded data. Normalization of data with such outliers leads to very small values for most data samples, which over-emphasizes the importance of the outliers compared to the rest of the data samples. To overcome this problem, sample values that exceed $\pm 6\sigma(x_i)$ were set to $\pm 6\sigma(x_i)$, rectifying outliers, where σ denotes the standard deviation for the EEG data of channel i.

5. **Normalization**
   After filtering and clipping, the EEG signals are normalized. For each session and electrode i the mean $\mu(x_i)$ of the signal is subtracted from every time measurement sample $x_i(t)$ and the result is divided by the standard deviation $\sigma(x_i)$ as shown in equation 3.1 below:

$$x_i^*(t) = \frac{x_i(t) - \mu(x_i)}{\sigma(x_i)} \tag{3.1}$$

3.2.2 Data Augmentation

Small data sets limit the ability of deep learning models to learn discriminative features and lead to overfitting. In other domains such as computer vision, data augmentation is common practice to enlarge the dataset. While methods such as stretching, compressing, rotating or flipping works well for image-like data, it is unsuitable for time-series data like EEG. Crops generated by sliding a fixed-size window over each EEG trial, however, has been shown to efficiently increase the amount of training examples, leading to a better performance of CNN models [60].

In this work, the crops are created using a sliding window with the length of 1024 timesteps, i.e., 4 seconds given the sampling frequency of 256 Hz. The sliding window shifts by n timesteps to create next crop until the end of the trial. Formally, given an original trial $\boldsymbol{X}^j \in \mathbb{R}^{\mathbb{E}\cdot\mathbb{T}}$ with E electrodes and T timesteps, the sliding window generates crops $\boldsymbol{C}^j$ with size T' as slices of the original trial as follows:

$$\boldsymbol{C}^j = \left\{ \boldsymbol{X}^j_{[1,E],[t,t+T']} | t \in [1, T - T'] \right\} \tag{3.2}$$

where j is the trial index.
This cropping strategy forces the deep learning model to learn discriminative features that are present in all crops of the trial because the model can no longer rely on the global structural differences between the original trials. Each crop receives the same label y_k as the original trial. Choosing a small value for the shift parameter n low leads to aggressive cropping, which in turn yields more but higher correlated new training examples. A shifting parameter of $n = f_s/8 = 32$ (i.e., 125 ms) yielded the best results regarding model performance in terms of accuracy. The crops were collected starting 3 seconds prior to motor imagery onset until the end of the trial, which guarantees that a minimum of 1 second motor imagery is present within the crop. In total, this cropping strategy yields 25 new examples per trial, that is, cropped training increases the training set by a factor of 25. In addition, as the sliding window is smaller than the trial, the input to the deep learning model is also smaller (2560 timesteps vs. 1024 timesteps for original and cropped trials, respectively).

3.2.3 Long Short-Term Memory Model

Given that neural activity measured by EEG can be regarded as highly dynamic, non-linear time series data, recurrent neural networks and particularly LSTM appear to be the tool of choice for modeling the temporal characteristics of brain activity. To assess the model's capability of learning discriminative features without prior manual feature-engineering, raw EEG timeseries with only minimal preprocessing (as explained in Section 3.2.1) is fed into the LSTM model.
The architecture of the proposed LSTM model is described in the following section.

Architecture

In this work, LSTM models with up to three LSTM layers consisting of 32 to 256 LSTM memory cells were regarded. Since it is not possible to know the best model architecture a priori, different configurations were trained and monitored on the validation set. The configuration performing best on the validation set was then evaluated on the test set. The best results have been achieved with a single layer and 128 LSTM memory cells, which is consistent with results obtained by Bashivan et al. who classified ERP signals [59]. More layers and a larger number of LSTM memory cells improved training accuracy, but led to strong overfitting and a performance decrease on the validation sets. Less than 128 memory cells led to underfitting, which was noticeable by a decrease in both training and validation accuracies. Downsampling the EEG data by a factor of two helped increase the performance of the model, as RNN models generally benefit from shorter time sequences.
The model is implemented as a many-to-one approach, i.e., only the prediction made by the LSTM after processing the whole sequence is propagated to the output layer. The output layer uses a softmax activation using one neuron per class. The continuous outputs of the softmax function are thresholded to obtain binary output predictions per training example.
The high number of parameters in the model make it prone to overfitting. To counter this problem, a dropout layer with a deactivation rate of 0.05 between output and LSTM layer has proven helpful. Recurrent dropouts, i.e., random deactivations of recurrent connections between timesteps drastically decrease the models learning ability and are therefore not recommended.
The LSTM model is recapitulated in Table 3.1.

Layer	**Input**	**Operation**	**Output**	**# Parameters**
1	$(T/2) \times E$	LSTM (128 hidden units)	128	67,584
	128	Dropout (0.05)	128	-
2	128	Softmax	K	258
Total				$67,842$

Table 3.1: Proposed LSTM architecture. E is the number of channels, T is the number of timesteps and K is the number of classes. Input and Output sizes are shown for cropped training with $E = 3$ (electrodes C3, C4 and Cz) and $T = 1024$ for window size of 4 seconds; binary classification with two classes for $K = 2$.

Training

The model was estimated using the fast and efficient Adam optimizer, a mini-batch gradient descent variant described in Section 2.2 with categorical cross-entropy (CE) as the loss-function (cf. Section 2.2). Optimizer parameters were set to a learning

rate of 10^{-3} and decay rates of first and second moments of 0.9 and 0.999, respectively (as recommended by the creators of Adam in their paper [27]). In order to speed up the training, the model was trained on the largest possible mini-batch sizes of 256 training examples, which is limited by the GPU video memory. Batch-wise updating of the parameters also makes the gradients less noisy and allows parallelization of computations on the GPU. The model was trained on a NVIDIA GTX Titan X GPU, with CUDA 8.0 and cuDNN v5, using Theano 0.9 [31] and Keras 2.0.5 API [34].
For each individual subject, training has been conducted using a *stratified 5-fold cross-validation*. More precisely, one of the five folds was held back for testing while the four remaining folds were used for training and validation with a split of 90 % and 10 %, respectively, in a loop until each fold has once been used for testing. *Stratified* in this context means that both classes are represented equally in each fold.

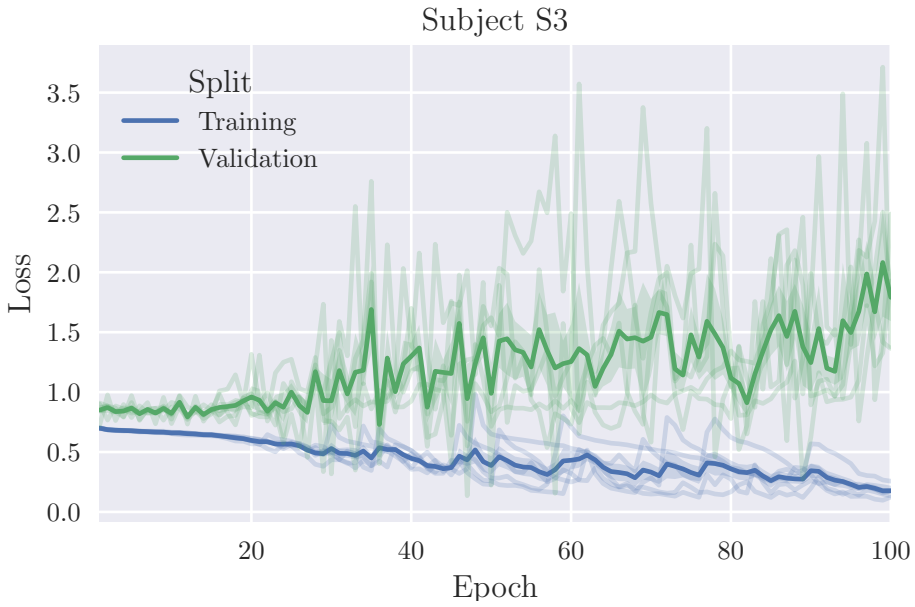

Figure 3.3: LSTM model optimization: Training and validation loss for all cross-validation folds for a single subject. Thick blue and green lines represent the average over all folds for training and validation, respectively.

Figures 3.3 and 3.4 visualize loss and accuracies over all cross-validation folds during training. A well-known pattern of overfitting is visible. Given that validation loss could not be further decreased using dropout or L2 regularization, another measure has been taken to counter overfitting: During the 100 training epochs validation loss was monitored as a metric for the model performance. Whenever an improvement in the validation loss was noticed a function was called back and a checkpoint of the model was created. At the end of training, the best model having the lowest

validation loss was restored and evaluated on the test set. The results are presented in a separate section in a later part of this thesis.

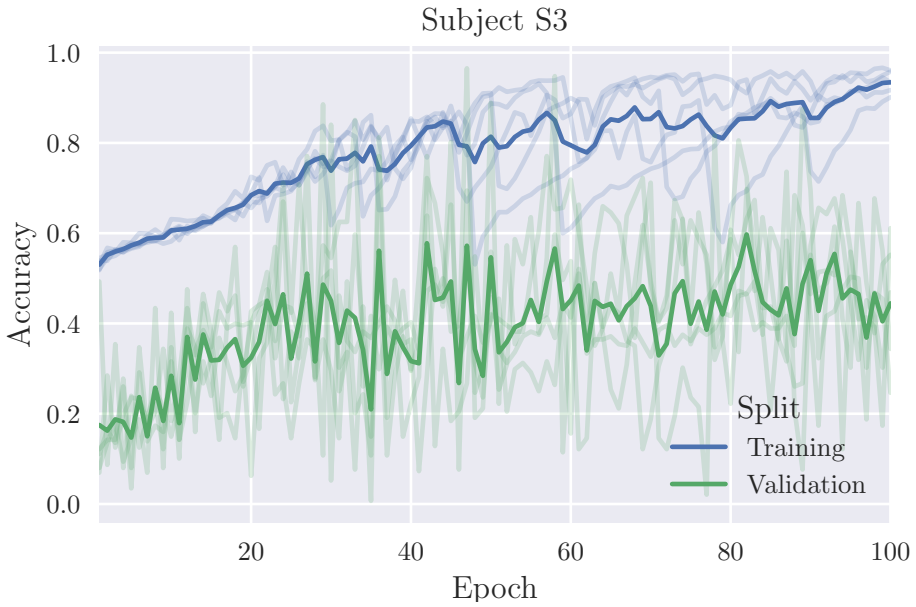

Figure 3.4: LSTM model optimization: Training and validation accuracies for all cross-validation folds for a single subject. Thick blue and green lines represent the average over all folds for training and validation, respectively.

Albeit its suitability for processing time series data, the proposed LSTM model did not perform as well as expected in classifying motor imagery EEG data. A reason for this might be the limited amount of data which rendered the use of more complex LSTM models with either more layers, more LSTM memory cells (or both) impractical. Therefore, another deep learning architecture has been evaluated and will be described in the following section.

3.2.4 Convolutional Neural Network Model

To date, convolutional neural networks are at the forefront of the most successful deep learning architectures when it comes to working with image-like data. This is mainly due to their ability to learn robust representations that are invariant against partial spatial translation or deformation. In the BCI community, some research on the use of CNNs on EEG data has already been conducted (cf. Bashivan et al. [59], Lawhern et al. [8], and more recently Schirrmeister et al. [60]), but most of the studies either attempt to use raw EEG data or classify other types of EEG signals, such as ERP or ERN.
This work aims to mimic the use of CNNs in computer vision by providing image-like data to the model in the form of spectrograms, which is a popular visualization

technique in audio signal processing. In the following sections, first the computation of spectrograms for EEG data is described and then the proposed architecture of the CNN model is explained.

Time-Frequency Representation of EEG data

One important constraint when transforming EEG data to an image-like representation is to retain both the time and the frequency information in the image. The most common method to achieve that in audio signal processing is the short-time Fourier transform (STFT). The signal is first cropped into short, overlapping time-windows. Then, a fast Fourier transform (FFT) is computed for each crop, assuming stationarity in short time-frames. The time-wise concatenation of the frequency components is also known as *spectrogram*, which represents the signal in both time and frequency. Figure 3.5 shows spectrograms of electrode channel C3 for both motor imagery classes, that is, left and right hand movements.

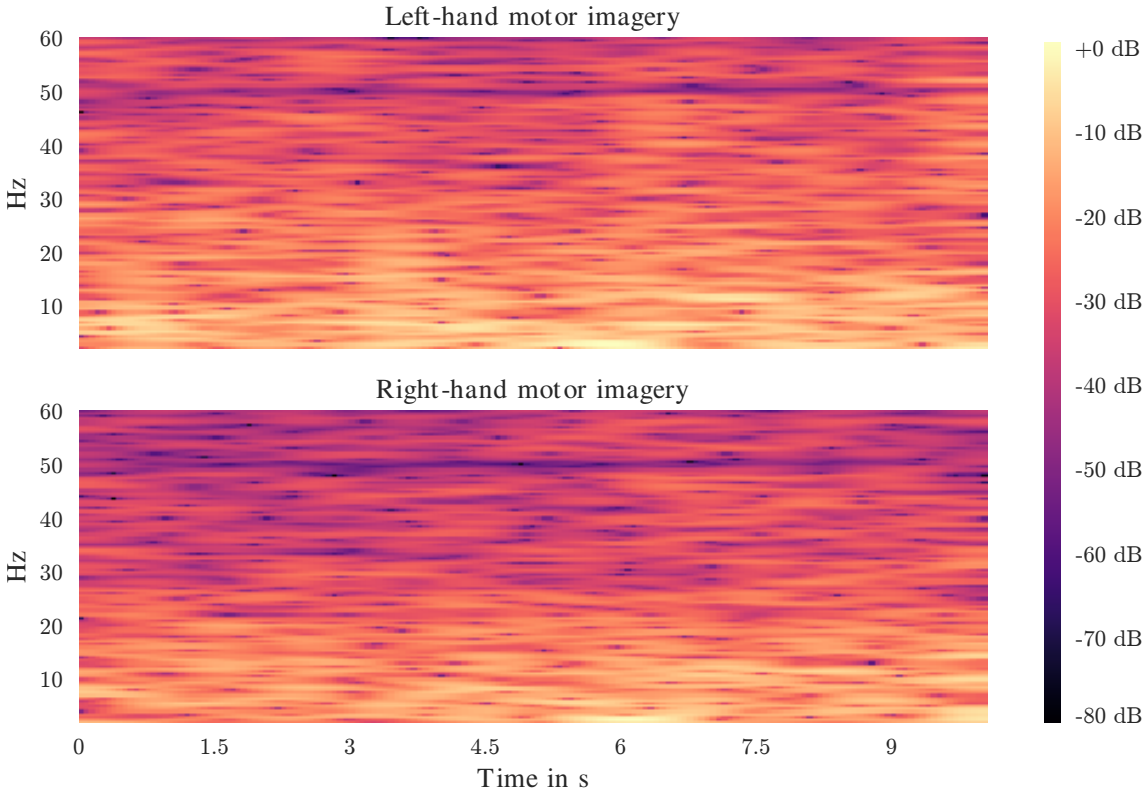

Figure 3.5: Spectrograms of electrode C3 of both classes for subject *S4*. Notice how especially the upper γ-band in the range between 25 to 40 Hz is visually different from one another for a human viewer. Parameters were set to $n = 1024$ FFT samples and a time shift of $s = 16$ time steps between STFT windows.

An important feature and a good sign of the potential use as an input to a CNN model is the apparent visual difference between the classes for a human observer.

In Figure 3.5, the most obvious differences are visible in the upper EEG frequency bands between 25 and 50 Hz.
In the proposed CNN model architecture, the spectrograms are computed on-the-fly using convolutional STFT kernels in a custom Keras layer on GPU with the library `Kapre` (Keras audio preprocessors) [61]. The model architecture is described in the subsequent section.

Architecture

The proposed CNN model is shown in Table 3.2 for EEG trials having E electrode channels and T time samples.

Layer	Input	Operation	Output	# Parameters
1	$E \times T$	STFT	$65 \times 64 \times E$	-
2	$65 \times 64 \times E$	$24 \times$ Conv2D (12×12)	$65 \times 64 \times 24$	10,392
	$65 \times 64 \times 24$	BatchNorm	$65 \times 64 \times 24$	260
	$65 \times 64 \times 24$	MaxPool2D (2×2)	$32 \times 32 \times 24$	-
	$32 \times 32 \times 24$	ReLU	$32 \times 32 \times 24$	-
	$32 \times 32 \times 24$	Dropout (0.5)	$32 \times 32 \times 24$	-
2	$32 \times 32 \times 24$	$48 \times$ Conv2D (8×8)	$32 \times 32 \times 48$	73,776
	$32 \times 32 \times 48$	BatchNorm	$32 \times 32 \times 48$	128
	$32 \times 32 \times 48$	MaxPool2D (2×2)	$16 \times 16 \times 48$	-
	$16 \times 16 \times 48$	ReLU	$16 \times 16 \times 48$	-
	$16 \times 16 \times 48$	Dropout (0.5)	$16 \times 16 \times 48$	-
3	$16 \times 16 \times 48$	$96 \times$ Conv2D (4×4)	$16 \times 16 \times 96$	73,824
	$16 \times 16 \times 96$	BatchNorm	$16 \times 16 \times 96$	64
	$16 \times 16 \times 96$	MaxPool2D (2×2)	$8 \times 8 \times 96$	-
	$8 \times 8 \times 96$	ReLU	$8 \times 8 \times 96$	-
	$8 \times 8 \times 96$	Dropout (0.5)	$8 \times 8 \times 96$	-
4	$8 \times 8 \times 96$	Flatten	6144	-
	6144	Softmax	K	12,290
Total				170,734

Table 3.2: Proposed CNN architecture. E is the number of channels, T is the number of timesteps and K is the number of classes. Input and Output sizes are shown for cropped training with $E = 3$ (electrodes C3, C4 and Cz) and $T = 1024$ for window size of 4 seconds; binary classification with two classes for $K = 2$.

The network architecture is inspired by CNNs used in ImageNet competition, such as VGGNet [62] and AlexNet [6]. It uses stacked convolutional layers with decreasing size and increasing number of filter kernels in deeper layers. After each convolutional layer, batch normalization is applied to reduce covariate shift in intermediate representations and improve robustness [50]. Then, max-pooling is performed, effectively

downsampling the output from the previous layer by a factor of two in the first two dimensions. A ReLU non-linearity is then applied, followed by a dropout with deactivation probability 0.5. The ensemble of convolutional layer, batch normalization, max-pooling, ReLU activation and dropout is repeated three times. Finally, the output of the last convolutional ensemble is flattened and fully-connected to a layer consisting of K neurons with softmax activation, where K is the number of classes. The input to the CNN model is raw EEG data with minimal preprocessing as described in Section 3.2.1. The actual spectrogram representation of the signal is computed in the first layer of the model, which is a custom Keras layer created by Keunwoo Choi [61]. The layer uses convolutional STFT kernels to compute the spectrogram representation for each of the EEG channels, stacking each of the spectrograms on top of each other. This approach keeps preprocessing to a minimum while exploiting the capabilities of parallelized GPU computation.
As with the LSTM model, other CNN configurations using different kernel sizes and numbers, the amount of stacked convolutional layers and the use of other dropout ratios have been tested. As there is a large number of tunable hyperparameters, a grid-search over all possibilities is extremely time-consuming. Therefore a random search has been performed, leading to the configuration shown in Table 3.2.

Training

The model was trained using the same training procedure as described for the LSTM model in Section 3.2.3.

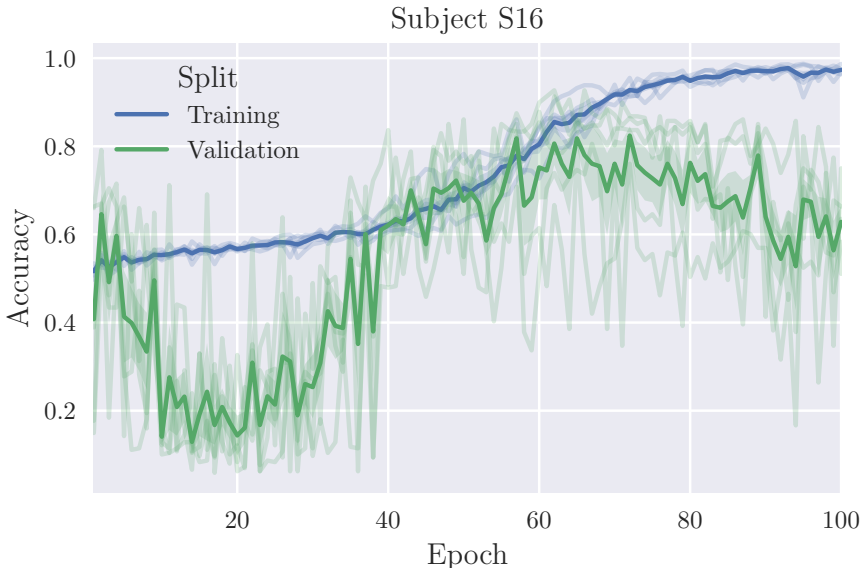

Figure 3.6: Spectrogram-based CNN model optimization: Training and validation accuracy for all cross-validation folds for a single subject. Thick blue and green lines represent the average over all folds for training and validation, respectively.

Figure 3.7: Spectrogram-based CNN model optimization: Training and validation loss for all cross-validation folds for a single subject. Thick blue and green lines represent the average over all folds for training and validation, respectively.

Figures 3.6 and 3.7 visualize accuracy and loss over all cross-validation folds during training for a single subject. It is noticeable that the CNN model converges with less fluctuations in both loss and accuracy when compared to the LSTM model. As with the LSTM model, overfitting is also an issue. In addition to dropout, the validation loss as a metric for the model performance is monitored over 100 training epochs. Whenever an improvement in the validation loss was noticed a function was called back and a checkpoint of the model was created. At the end of training, the best model having the lowest validation loss was restored and evaluated on the test set. The results are presented in Section 3.3.

3.3 Results

This section shows the results obtained with the proposed LSTM and CNN models for 20 subjects from the NST dataset. As a validation baseline, an additional subject from the Graz B1 dataset has been included. Furthermore, shallow and deep CNN models that were recently published (see Section 2.4) have been reimplemented and evaluated on both NST and Graz data. A discussion of the results is presented in Chapter 4.

Accuracy was used to evaluate the model's performance. For each individual subject, training and testing have been conducted using a *stratified 5-fold cross-validation*.

Test sets of all subjects were chosen to be identical for each model to ensure comparability. The test results over all cross-validation folds are presented in the form of *boxplots* for all models: The bottom and top of the box are the first and third quartiles, and the band inside the box is the second quartile (the median). The whiskers represent minimum and maximum. Any data point not included between the whiskers is plotted with a dot, representing outliers.

3.3.1 Proposed Long Short-Term Memory Model

Figure 3.8 presents the results obtained with the proposed LSTM model.

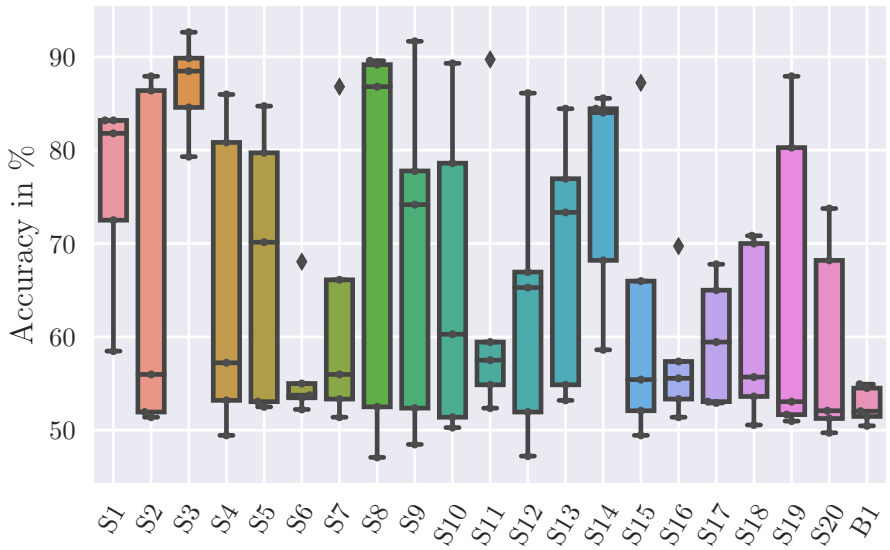

Figure 3.8: Boxplot showing within-subject accuracies for the proposed LSTM model trained on augmented data with 5-fold cross-validation.

The overall mean accuracy over all subjects is 66.20 % (±7.21%), which is well above the chance level for binary classification. For the best performing subject S3, the mean accuracy reached 86.97% (±5.18%). In contrast to that, for many subjects the obtained accuracies were only slightly higher than the chance level (below 60% for S6, S16, S17, S18 and S20). Graz B1 also performed not significantly above chance level with an accuracy of 52.68 %. Overall, the LSTM model performs comparable to state-of-the-art models using classical machine learning classifiers, such as support vector machines (66.9 % in [2]) in terms of accuracy. However, it is very unstable across the test splits, which is shown by the large variance in the accuracy for within-subject test splits.

3.3.2 Proposed Convolutional Neural Network Model

Figure 3.9 presents the results obtained with the proposed CNN model.
The overall mean accuracy over all subjects is 84.24% (±14.69%). The standard deviation (SD) in the mean accuracy between subjects can be mainly attributed to the inability of the proposed CNN model to classify data from subjects S5, S10 and 19, which reached accuracies close to the chance level, with a minimum as low as 50.08% for subject S19. In general, the CNN model is very stable across test splits with an average SD of 3.32% across all subjects, with S6 being an exception (SD of 15.13%).

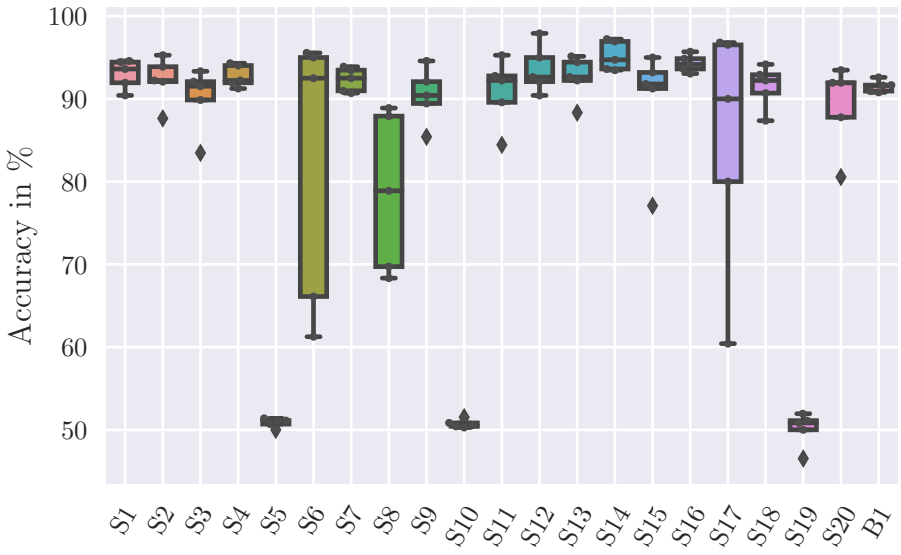

Figure 3.9: Boxplot showing within-subject accuracies for the proposed CNN model trained on augmented data with 5-fold cross-validation.

3.3.3 State-of-the-Art CNN Models

The following two sections present the results achieved with two CNN models recently published by Schirrmeister et al. [60]. They were reimplemented and tested in this work under the same conditions as the proposed models in the previous sections (see Section 2.3.2 for details on model architecture).

Shallow Convolutional Neural Network Model

Figure 3.10 presents the results obtained with the shallow CNN model.
The overall accuracy over all subjects is 66.97 (±6.45%), which is slightly worse than the results achieved by its authors (71.9% in [60]). For the best performing

subject S17, the mean accuracy reached 78.89% (±7.77%). As for the proposed LSTM model, some subjects reached mean accuracies only slightly above chance

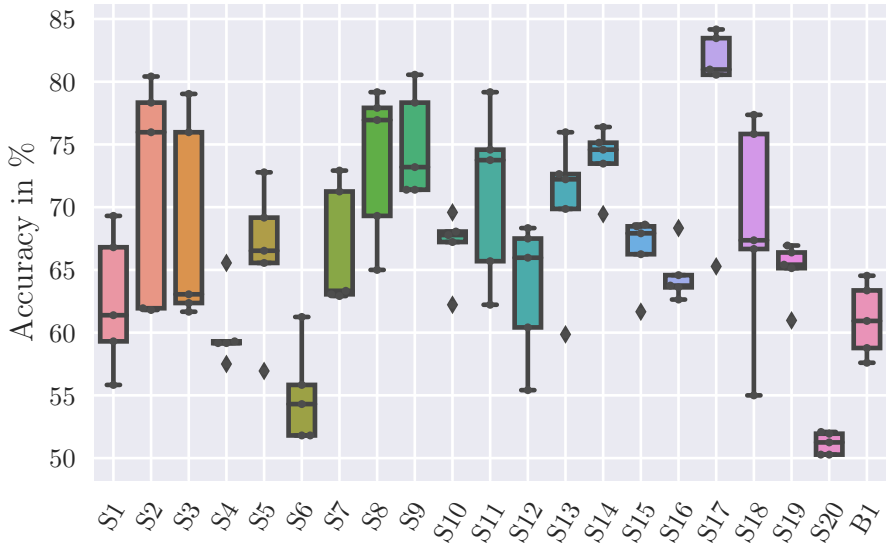

Figure 3.10: Boxplot showing within-subject accuracies for the Shallow CNN model on augmented data with 5-fold cross-validation.

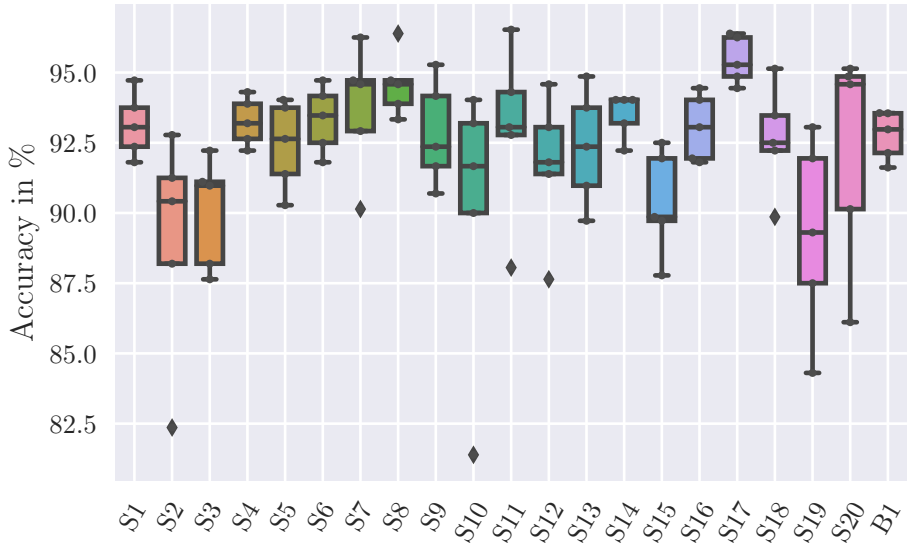

Figure 3.11: Boxplot showing within-subject accuracies for the Deep CNN model on augmented data with 5-fold cross-validation.

level (below 60% for S4, S6 and S20). Graz B1 also performed not significantly above chance level with an accuracy of 52.68 %. Overall, the shallow CNN model performs comparably to the proposed LSTM model. However, it is more stable across the test splits, which is shown by a smaller variance in accuracy for within-subject test splits when compared to the LSTM model.

Deep Convolutional Neural Network Model

Figure 3.11 presents the results obtained with the deep CNN model. The overall mean accuracy over all subjects is 92.29% (±1.69%). For the best performing subject S17, the mean accuracy reached 95.44% (±0.85%). In contrast to other evaluated models in this work, the deep CNN models was able to accurately classify data within all subjects, as the lowest mean accuracy reached still was relatively high at 89.0% (±4.06%). Graz B1 performed comparably to NST subjects with a mean accuracy of 92.77% (±0.87%). Furthermore, the deep CNN model shows particularly stable results across within-subject test splits with a SD of 2.16%.

3.3.4 Summary

Figure 3.12 shows a summary of the results obtained with the proposed LSTM and CNN models, as well as deep and shallow CNN models, representing the state of the art in deep learning for BCI tasks. First, it can be seen that the proposed LSTM model was on a par with the shallow CNN model for most subjects, albeit being less stable, which can be seen by a considerably higher standard deviation. An exception is subject S17, where the shallow CNN outperformed the LSTM model by a large margin ($\approx 20\%$).

The proposed CNN model achieved slightly lower accuracies than the deep CNN model for most subjects. While it excels for six subjects (S2, S6, S13, S15, S18 and S20), it shows inferior performance for 13 subjects (S1, S3, S5, S7, S8, S9, S10, S11, S12, S14, S16, S17 and S19). Additionally, the proposed CNN model was unable to classify data from subjects S5, S10 and S19, showing near chance-level performance. On the baseline test set of subject B1 from Graz data set, both models achieve comparable performance (91.52% vs. 92.77%).

The results are compared and discussed in the following chapter.

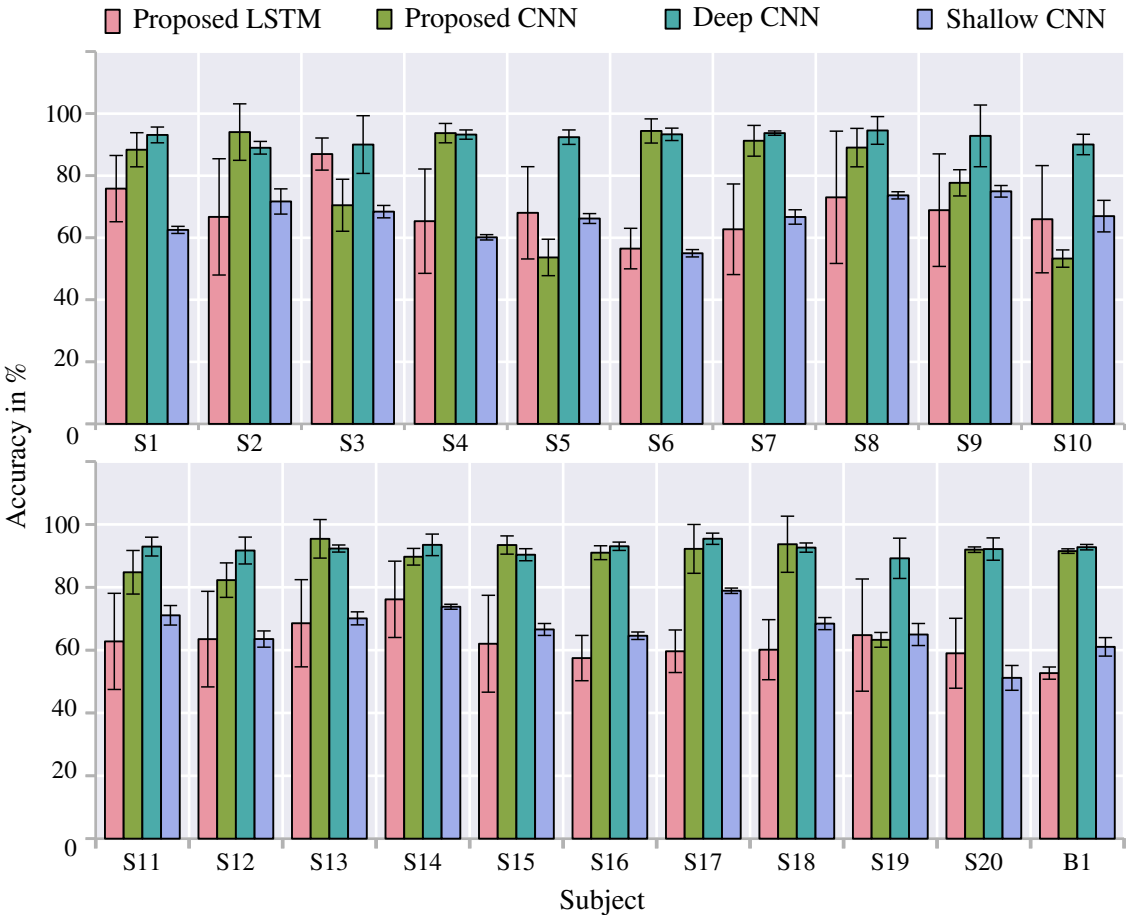

Figure 3.12: Summary of the performance for all implemented models. The height of the bars represents the mean accuracy over 5-fold cross-validation on tests sets; whiskers represent the standard deviation.

Chapter 4

Discussion

In this work different deep learning models were developed to classify SMR features in EEG data for the use in a BCI: a LSTM model, a spectrogram-based CNN model, and deep and shallow CNN models proposed in a recent publication [60]. Table 4.1 shows an overview of the results of this work compared to similar approaches in literature.

Author	**Reference**	**Model**	**Accuracy ($\pm\sigma$)**	**Classes**
This work	Section 3.2.3	Proposed LSTM	66.20% (±7.21%)	2
This work	Section 3.2.4	Proposed CNN	84.23% (±14.69%)	2
This work	Section 2.4	Shallow CNN	66.97% (±6.45%)	2
This work	Section 2.4	Deep CNN	92.28% (±1.69%)	2
Schirrmeister et al.	[60]	Shallow CNN	71.90% (± n.a.)	4
Schirrmeister et al.	[60]	Deep CNN	70.10% (± n.a.)	4
Yang et al.	[56]	FBCSP	67.01% (±16.20%)	4
Yong et al.	[2]	SVM	66.90% (± n.a.)	2

Table 4.1: Comparison of classification results for motor imagery tasks using deep learning models and traditional machine learning methods.

The LSTM model shows a performance comparable to the SVM model developed by Yong et al. using ERD/ERS, falling short by only 0.7% mean accuracy. Considering its suitability for processing time series data, the proposed LSTM model did not perform as well as expected in classifying motor imagery EEG data (see Section 3.2.3 for details). A reason for this might be the limited amount of data, which rendered the use of more complex LSTM models with either more layers, more LSTM memory cells (or both) impractical. A preliminary test using STFT time-frequency input to the LSTM did not show an improvement but a drop in performance, which is why an alternative approach using CNNs was developed.

The proposed CNN model using time-frequency representations of the EEG input improved the mean accuracy by a large margin of 18.02%. Considering the higher number of tunable parameters for the proposed CNN model (170,734 vs. 67,84), this

is an interesting result. Theoretically, a higher number of parameters requires more training data to avoid overfitting. However, the proposed CNN model shows less fluctuations during training in both training and validation loss. This leads to the conclusion that it is not straight-forward for a LSTM to learn relevant features from EEG data. The fact that the deep CNN model implemented in this work achieves the highest overall accuracy albeit using raw EEG input, supports the assumption that CNN architectures are inherently better suited to extract discriminative features from EEG. However, CNNs typically work with fixed-size inputs, whereas RNNs are able to handle sequences of arbitrary lengths. The fixed input size necessarily introduces a time delay in an online BCI setup, which makes RNN-based approaches, e.g., using LSTM, generally favorable.

Interestingly, in the original work of Schirrmeister et al. [60] the shallow CNN model achieved slightly better results than the deep CNN model (71.90% vs. 70.10% for a four-class problem) on the publicly available BCI Competition IVa dataset. When using an in-house recorded dataset (High-Gamma Dataset, see original publication for details [60]), the results shift in favor of the deep CNN model (92.5% vs. 89.3%). The superiority of the deep CNN model is supported by the results obtained in the implementation of both models in this work (92.28% vs. 66.97%). However, the drop in accuracy between the two CNN models in this work is striking. The reasons for this are unclear, but a possible explanation could be differing parameters for the Adam optimizer, as they were not described in the original publication of the models. Better hyperparameters for the optimizer could significantly improve training, which holds especially true for models that use non-standard activation functions such as squaring or taking the logarithm, which were both utilized in the shallow CNN model.

Chapter 5

Conclusion

This work contributed two deep learning-based models for decoding motor imagery movements from EEG data to the BCI toolbox. It was shown that a simple LSTM model can successfully learn and classify EEG data in a two-class classification task. The obtained accuracy is comparable to state-of-the-art results using traditional methods, but inferior to the outcome of CNN models.

Initially, this thesis project intended to implement the LSTM model for a four-class classification task and then convert it to a spiking neural network architecture for the use on the neuromorphic platform TrueNorth [63]. However, the LSTM model proved to be inferior to CNN-based approaches, which is why the objective of this work was reshaped in favor of the development of a CNN model. In retrospect, this decision proved to be favorable, as the proposed CNN model possesses several advantages over the LSTM model. First, the performance of the CNN model is vastly superior to the LSTM model. In addition, preliminary tests on a three-class classification task on the NST data showed promising results with only small drops in accuracy compared to the respective two-class problem. Second, and more importantly, IBM provides a software framework for training and deployment of CNN-based architectures, while RNN-based implementations on TrueNorth are still in its infancy [64]. Currently, a CNN model on TrueNorth only supports quadratic, low-resolution input. Therefore, a time-frequency transformation using STFT is particularly suitable for mapping relevant channels of high-dimensional EEG data to lower-dimension spectrogram representations. Therefore, the proposed CNN model has a better prospect concerning its convertibility to TrueNorth than the deep CNN model, which uses raw EEG input.

All of the models presented in this work still leave room for considerable improvement, especially for multiclass classification tasks. Bashivan et al. combined both CNN and LSTM to a hybrid model that outperformed individual CNN and LSTM implementations on event-related potentials [59]. A similar approach has not yet been evaluated on motor imagery tasks, but it seems extremely promising as it could combine both the capability of convolutional kernels to extract robust features with

the ability of RNNs to process sequential data.

As limited EEG data availability poses a problem for deep learning approaches, *transfer learning* could turn out to be the redemption for stagnating decoding accuracies in BCI research. Transfer learning is based on the idea that knowledge discovered in solving one task could also be useful for another, related task. For the case of EEG classification, a related field could be audio signal processing. Large-scale datasets of labeled audio events are publicly available (e.g., Audio Set by Goolge Research [65]), which could be used for pre-training deep learning models. The pre-trained layers could then serve as feature extractors in another model for EEG classification, as models trained on audio data are typically able to extract information from frequency domain. As their sampling rate and frequency range differs from EEG data, audio signals could be pitch-shifted and resampled to better match the characteristics of EEG recordings.

List of Figures

Bibliography

[1] Niels Birbaumer. Brain–computer-interface research: coming of age, 2006.

[2] Xinyi Yong and Carlo Menon. EEG classification of different imaginary movements within the same limb. *PloS one*, 10(4):e0121896, 2015.

[3] R Leeb, C Brunner, G Müller-Putz, A Schlögl, and G Pfurtscheller. BCI Competition 2008–Graz data set B. *Graz University of Technology, Austria*, 2008.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[5] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] Elliott M Forney. *Electroencephalogram classification by forecasting with recurrent neural networks*. PhD thesis, Colorado State University, 2011.

[8] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. EEGNet: A compact convolutional network for EEG-based brain-computer interfaces. *arXiv preprint arXiv:1611.08024*, 2016.

[9] Reza Fazel-Rezai, Brendan Z Allison, Christoph Guger, Eric W Sellers, Sonja C Kleih, and Andrea Kübler. P300 brain computer interface: current challenges and emerging trends. *Frontiers in neuroengineering*, 5, 2012.

[10] Gert Pfurtscheller and Christa Neuper. Motor imagery and direct brain-computer communication. *Proceedings of the IEEE*, 89(7):1123–1134, 2001.

[11] Martin Lotze, Pedro Montoya, Michael Erb, Ernst Hülsmann, Herta Flor, Uwe Klose, Niels Birbaumer, and Wolfgang Grodd. Activation of cortical and cerebellar motor areas during executed and imagined hand movements: an fMRI study. *Journal of cognitive neuroscience*, 11(5):491–501, 1999.

[12] Jean Decety. The neurophysiological basis of motor imagery. *Behavioural brain research*, 77(1):45–52, 1996.

[13] Robert Leeb, Felix Lee, Claudia Keinrath, Reinhold Scherer, Horst Bischof, and Gert Pfurtscheller. Brain–computer communication: motivation, aim, and impact of exploring a virtual apartment. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 15(4):473–482, 2007.

[14] Richard W Homan, John Herman, and Phillip Purdy. Cerebral location of international 10–20 system electrode placement. *Electroencephalography and clinical neurophysiology*, 66(4):376–382, 1987.

[15] John J Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, 4(5):3–10, 1988.

[16] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[17] Giambattista Parascandolo, Heikki Huttunen, and Tuomas Virtanen. Recurrent neural networks for polyphonic sound event detection in real life recordings. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 6440–6444. IEEE, 2016.

[18] Michael A Nielsen. Neural networks and deep learning, 2015.

[19] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[20] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[21] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[22] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220, 2010.

[23] Fei-Fei Li and Andrej Karpathy. Convolutional neural networks for visual recognition, 2015.

[24] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[25] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[26] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[27] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, pages 1–17, 2017.

[29] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards AI. *Large-scale kernel machines*, 34(5):1–41, 2007.

[30] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[31] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint*, 2016.

[32] Dong Yu and Xuedong Huang. Microsoft computational network toolkit (CNTK). 2015. *A Tutorial Given at NIPS*, 2015.

[33] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[34] François Chollet et al. Keras, 2015.

[35] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[36] Alan Julian Izenman. Linear discriminant analysis. In *Modern multivariate statistical techniques*, pages 237–280. Springer, 2013.

[37] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[38] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1):55–63, 1968.

[39] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[40] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

[41] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Communications of the ACM*, 54(10):95–103, 2011.

[42] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[44] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[45] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[46] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[47] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2012.

[48] Christopher Olah. Understanding LSTM networks. Technical report, August 2015. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[49] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. 1999.

[50] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[51] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[52] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[53] Abdulhamit Subasi and M Ismail Gursoy. EEG signal classification using PCA, ICA, LDA and support vector machines. *Expert Systems with Applications*, 37(12):8659–8666, 2010.

[54] Yijun Wang, Bo Hong, Xiaorong Gao, and Shangkai Gao. Implementation of a brain-computer interface based on three states of motor imagery. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 5059–5062. IEEE, 2007.

[55] Deon Garrett, David A Peterson, Charles W Anderson, and Michael H Thaut. Comparison of linear, nonlinear, and feature selection methods for EEG signal classification. *IEEE Transactions on neural systems and rehabilitation engineering*, 11(2):141–144, 2003.

[56] Huijuan Yang, Siavash Sakhavi, Kai Keng Ang, and Cuntai Guan. On the use of convolutional neural networks and augmented CSP features for multi-class motor imagery of EEG signals classification. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 2620–2623. IEEE, 2015.

[57] Sebastian Stober, Avital Sternin, Adrian M Owen, and Jessica A Grahn. Deep feature learning for EEG recordings. *arXiv preprint arXiv:1511.04306*, 2015.

[58] Alex S Greaves. Classification of EEG with recurrent neural networks. 2014.

[59] Pouya Bashivan, Irina Rish, Mohammed Yeasin, and Noel Codella. Learning representations from EEG with deep recurrent-convolutional neural networks. *arXiv preprint arXiv:1511.06448*, 2015.

[60] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep learning with convolutional neural networks for EEG decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420, 2017.

[61] Keunwoo Choi, Deokjin Joo, and Juho Kim. Kapre: On-GPU audio preprocessing layers for a quick implementation of deep neural network models with keras. In *Machine Learning for Music Discovery Workshop at 34th International Conference on Machine Learning*. ICML, 2017.

[62] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[63] Ewan Nurse, Benjamin S Mashford, Antonio Jimeno Yepes, Isabell Kiral-Kornek, Stefan Harrer, and Dean R Freestone. Decoding EEG and LFP signals using deep learning: heading TrueNorth. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 259–266. ACM, 2016.

[64] Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, David P Widemann, Adam T Moody, Brian C Van Essen, and Qinru Qiu. A spike-based long short-term memory on a neurosynaptic processor.

[65] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *IEEE ICASSP*, 2017.

License