Methods for Multivariate Time-Series Classification on Brain Data

Aggregation, Stratification and Neural Network Models

Dissertation

zur Erlangung des akademischen Grades
des Doktors der Naturwissenschaften (Dr.rer.nat.)

vorgelegt von

Christoph Doell

an der Universität Konstanz

Faculty of Science
Department of Computer and Information Science
Chair for Bioinformatics and Information Mining

Konstanz, 2020

Tag der mündlichen Prüfung: 08.03.2021

1. Referent: Christian Borgelt

2. Referent: Michael Berthold

Zusammenfassung

Die Arbeit beschäftigt sich mit der Analyse von Hirnstromdaten, gemessen mit Hilfe von Elektroencephalographie (EEG). Der erste Datensatz enthält Daten von Rauchern, die mehrere Stunden nicht geraucht haben, von Rauchern, die kurz vor der Messung geraucht haben und von Nichtrauchern. Diese Klassen sollen mit Hilfe von neuronalen Netzen unterschieden werden. Der zweite Datensatz enthält Hirnsignale und verschiedene Formen von Rauschen und Störsignalen und unverrauschte Signale, die voneinander unterschieden werden sollen. Um sie zu analysieren adaptiere ich eine Netzwerkstruktur, die für Neuronale Netze zur Objekterkennung in Bildern entwickelt wurde. Ich passe die sogenannten *residual blocks* an, um sie auf EEG-Zeitreihen anzuwenden. Eine Schwierigkeit von EEG-Daten ist die Tatsache, dass diese Daten personenbezogen sind. Diese kann allerdings manchmal sogar hilfreich sein, um verbesserte Vorhersagen zu machen: wenn sich die Klassen in den Daten selten genug ändern, dass davon ausgegangen werden kann, dass mehrere Stücke (sogenannte *Schnipsel*) eines längeren Signals zur selben Klasse gehören, dann kann dies verwendet werden, indem die Vorhersagen mehrerer Schnipsel zu einer Gesamtvorhersage zusammengefasst werden.

Ich untersuche insgesamt 15 Forschungsfragen aus den Bereichen der Neurowissenschaft, Anpassungen und Verbesserungen von neuronalen Netzen, und bezüglich der optimalen Wahl der Aggregationsfunktion, die zum Zusammenfassen der Schnipsel verwendet wird.

Abstract

This thesis is about the analysis of two data sets consisting of human brain data measured by electroencephalography (EEG). One data set contains data from craving smokers, who have not smoked for several hours, non-craving smokers, who had a smoke shortly before the measurement and non-smokers. These classes are to be distinguished with the help of neural networks. The second data set contains noisy EEG signals with different kinds of noise from and clean signal that are to be distinguished. In order to analyze them, I adapt a network structure, that was originally developed for neural networks for object recognition in images. I modify, the so called *residual blocks* in order to use them on EEG time series. One difficulty of EEG data is their property of being individual-specific. This can sometimes even be helpful to get improved predictions: if the classes of the data change so infrequently that it can be assumed that several parts (so called *snippets*) of a longer signal belong to the same class, then this information can be used to make predictions of several snippets and aggregate them to create a classification of the longer original signal.

I investigate a total of 15 research questions, regarding the context in neuroscience, adaptations and improvements of neural network models, and the optimal choice of aggregation functions.

Acknowledgements

I want to thank my advisors, Prof. Dr. Christian Borgelt and Prof. Dr. Michael Berthold as well as my former advisor Prof. Dr. Rudolf Kruse. Thank you very much for your time and your support. You have taught me how to work scientifically, how to give proper presentations and also shared your life experience, which was maybe even more helpful. Special thanks to Christian Borgelt, who kept asking questions regarding those small details that are too frequently taken as given. I think these questions and your patience in combination with your seemingly never-ending will to educate, made the past years for me not just exhausting, but also exciting and interesting. I enjoyed working with all of you.

The professors enable to build workgroups, but they finally consist of many people. I had the opportunity to be part of two great workgroups: I enjoyed a great time with my colleagues and friends Dr. Christian Braune, Peter Burger, Dr. Alexander Dockhorn, Dr. Alexander Fillbrunn, Heather Fyson, Pascal Held, Dr. Martin Horn, Katrin Krieger, Sabine Laube, Dr. Oliver Sampson, Dr. Patrick Winter and all the other nice people I met along my path in Mainz, Magdeburg and Konstanz.

Finally, I want to thank my family, my friends and my girlfriend Beate Krestel, and last but not least Maximilian von Platen for reading every word of every sentence of this work, asking about the details, discussing them with me, forgetting them, reading them again, discussing them again, and finally fighting about the best fitting words, and then forgetting them again.

Without all your support and help this work would not have been possible.

I have (co-)authored the nine publications listed below. This thesis is mostly based on the first three, which focus on aggregation methods or a data set aiming on differentiating smokers and non-smokers:

1) 'Aggregation of Subclassifications: Methods, Tools and Experiments' [34] (Chapter 4)
2) 'Training Neural Networks to Distinguish Craving Smokers, Non-craving Smokers, and Non-smokers' [38] (Chapter 5)
3) 'Residual Neural Networks to Distinguish Craving Smokers, Non-craving Smokers and Non-smokers by Their EEG Signals' [35] (Chapter 5)

I have also worked on machine learning in games. This field uses reinforcement learning instead of classical supervised learning to train agents. In the first work, I used a *wide* and a *deep* component to train PacMan to win in this classic Atari game. In the second one, I used Monte Carlo Tree Search to train an agent playing the card game doppelkopf.

4) 'Wide and Deep Reinforcement Learning Extended for Grid-Based Action Games' [101]
5) 'A decision heuristic for Monte Carlo tree search doppelkopf agents' [32]

Further, I published a work on cognitive architectures. These algorithms are created to model 'human-like' thinking, with humans as role-model for general intelligence. One goal of this field is to create algorithms that generally perform well (artificial general intelligence) as opposed to well-known algorithms that only perform well on very particular tasks. Another goal is to create computational models (cognitive architectures) of the human mind aiming for a better understanding of how humans think.

6) In 'Evaluation of cognitive architectures inspired by cognitive biases' [36] I developed a measure of how 'human-like' an algorithm acts by analyzing which human-like mistakes the algorithm makes.

The following two publications try to prevent future major-accidents by analyzing human interactions of such accidents of the past. I utilized association rule mining and hierarchical clustering to find common patterns there.

7) 'A clustering approach to a major-accident data set: Analysis of key interactions to minimise human errors' [102]
8) 'Analysis of a major-accident dataset by Association Rule Mining to minimise unsafe interfaces' [37]

Finally, I worked on graph theory, looking for farthest points in networks:

9) 'Network farthest-point diagrams' [54]

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

This work is based on two major motivations, one stemming from the biological side of getting a better understanding of addiction and one stemming from recent progress in the field of neural networks. The work of Attia [5] is an example, where data from electrocardiography (measuring heart beats) is passed to a neural network, which is able to use them to predict the age of a person with surprising accuracy. It shows the power of neural networks to detect features in data, and that biological time series data may contain surprising information on the subject.

The use of drugs, whether legal or illegal, has a profound impact on the brain. Not only do drugs act on neuronal receptors, causing changes in the signaling patterns of these cells, they also induce a sensation of craving when they are gone, due to the effects of withdrawal. Understanding how addiction works in the brain is therefore of utmost importance, as it is the first step in determining the best ways to treat addiction. Nicotine is legally used worldwide and provides an excellent opportunity to study addiction in the brain for multiple reasons. First, nicotine, like other drugs, has chemical effects on the brain, which can be measured (e.g., [91]). Second, after only few hours of abstinence, smokers start to crave the next cigarette — a hallmark of addiction, the neural underpinnings of which are little understood. Third, the legality and prevalence of nicotine provides an available subject population, without the ethically and legally questionable issues that can be present when examining addiction in illegal substance abuse. Fourth, the study of addiction in humans avoids the ethically questionable administration of drugs to animals, which may or may not respond in similar ways to the drugs as humans do [93, 119]. However, the legality of some drugs, such as nicotine, makes it possible to study addiction in humans (more details on addiction are given in Chapter 3), and non-invasive measures such as electroencephalography (EEG), provide a time-point-by-time-point measure of the neural signal. Understanding how these EEG signals may differ between non-smokers, sated smokers (non-craving), and craving smokers could help shed new light on the changes of brain function in addiction.

One promising way to examine such differences is to apply recent advancements in machine learning algorithms to accurately classify these data into the categories of craving smokers, non-craving smokers, and non-smokers.

As nicotine influences neural functioning, this should manifest in some alteration of signaling that can be measured with EEG. Moreover, the brain-state of craving would be expected to have different (and possibly more salient) patterns than non-craving, due to the stress/arousal present with the desire to smoke. Experts in the field believed this to be impossible. However, my research in Chapter 5 proves that results better than random guessing are possible.

I am specifically interested in the differences in the resting state data, that are acquired over a period of time during which the participant has no specific task to do. The idea is to monitor natural fluctuations in activity that are *not* due to the onset of an image or sound. Any neural activity that is a result of the use of nicotine, and therefore common across the smokers, should be present in the resting state, and not masked by any task. This pattern of activity may be subtle, however, and difficult to extract without knowing what the specific patterns are. As such, neural network models may be able to find patterns and classify data in a way that more traditional cognitive neuroscience-based analyses cannot. Neural networks, with their ability to automatically identify and learn important features and with their recent advances like convolutions with skip connections, could be well-suited for this task.

Recent years have brought huge advances in image processing with the help of neural networks. While in 1990 it was still very difficult to automatically detect whether or not a picture contained a human face, today it is possible to *recognize a person* by their face on a picture automatically. These advancements were not only caused by improved hardware, which performs many computations in parallel on a GPU, but also by improved algorithms.

One of the more important additions to neural networks are convolutional layers. They move a filter over the image that detects patterns independent of their location. Before their introduction, neural networks had major problems with detecting objects in images consistently. Even though the neural network may have learned to identify an object within images of the training data, it could not transfer this information to new data, where the same object had a slightly different position within an image.

With improved hardware it became feasible to build bigger neural networks. Although bigger networks would in principle be more accurate, they often could not be trained properly. This is where the other big improvements were made: Different activation functions were used, the initialization of the weights was improved, optimizers were created that automatically regularize the training speed of the parameters and better network structures were introduced in the form of skip connections. More details of how this works, and how I adapted the ideas that were introduced for pictures to EEG data can be found in Section 2.4.

In Section 3.3 I give examples of applications of neural networks from different fields of research. Some fields showed great improvements since neural networks were first applied. My inspection shows similarities between these

areas and finally compares them to the field of EEG analysis, highlighting its inherent difficulties.

How EEG data are taken, which difficulties (like different sampling rates and several kinds of noise) accompany these measurements, and how preprocessing approaches like resampling or an Independent Component Analysis try to cope with them are described in Chapter 2. One of the obstacles that comes with the analysis of EEG data is that all measurements are individual-specific and show a greater variability across subjects. Also, there is no human sense that enables to directly interpret this kind of data. These facts, combined with the normally small amount of measurements, makes it hard to distinguish individual-specific effects from those that are associated with the attribute one is interested in (e.g. the smoker status).

Different measurements from the same subject must be considered statistically dependent. This increases the difficulty of the validation process, since this process must guarantee an independent split between the training and the testing set. Algorithms can cope with that if the given data set contains similar amounts of samples per class. Otherwise, stratification techniques need to be applied, which, in combination with the statistically dependence, make it (NP-)hard to find a proper split, as my research in Section 2.3.2 shows.

However, in some special cases, the statistical dependence can also be used to *improve* the prediction quality by the help of the following procedure: If all data from one measurement or subject belong to the same (unknown) class, this information can be used to split the measurement into smaller pieces, make predictions for these small pieces and then combine them into a prediction for the whole measurement with the help of an aggregation function. This is why in this work I am mostly interested in data with constant states, which change slow enough to observe only one class during one measurement. Unfortunately, I did not find any other available EEG data set other than the smoker data set from Chapter 5 that fulfilled this criterion.

Note that this procedure is not only possible for biological time series, but can be used in any case, where a classification is to be performed on objects that are too big. Even if the original objects do not fit into the main memory as a whole, smaller sub-objects can be constructed that fit.

This leads to my research questions on how this aggregation should be applied in order to achieve the optimal classification. Although the aggregation of features and the aggregation of classifier results was investigated before, there are only few works that investigate the effects of different aggregation functions in practice. Of all possible aggregation functions, one may yield better results than the others, or maybe the best function depends on some characteristics of the underlying data.

When trying to figure out how the distribution of the data influences the quality of the aggregation functions, I need to be able to manipulate that distribution. Therefore I cannot use any fixed (real world) data set, but I

need to simulate data with different simulation parameters. Depending on those parameters, I evaluate six basic aggregation function, to find the best one in Chapter 4. Further, the simulated data must be biased toward the true class, for which I investigate two different types and several intensities. Since there are many more possibilities for aggregation functions and the modeling of bias, researchers are invited to include them into my publicly available source code and use it to investigate their properties.

Chapter 5 considers the smoker data set, which allows to apply aggregations, since the status of being a smoker or non-smoker does not change too frequently. EEG measurements are relatively long, and only few of them are available. This is difficult for classification algorithms, since good classifications need as many data samples as possible, and each sample should have a sufficiently small dimensionality. In order to minimize this problem, I apply a bootstrapping method that split the data into several smaller parts, I call *snippets*. Those snippets perfectly fulfill the criteria needed to apply aggregation functions, such that this chapter is used to validate the theoretical results obtained in Chapter 4 in practice.

Another group of my research questions focuses on neural networks. Since many of the accomplishments and lots of studies in that field focus on pictures as input data, I modify and apply some of these methods to EEG data, which I use in form of multivariate time-series. Is it also possible to improve the neural network's performance when processing EEG data, with residual blocks? A convolution that is applied on each channel individually might help to rescale the channels and thus might improve the training process.

The third area of research I am interested in, is the neuro-scientific part: Is it possible to distinguish smokers from non-smokers by their EEG data? Can the craving status be distinguished? Are there significant differences in the prediction qualities depending on the subject? Is the network able to successfully find patterns that can be applied to other subjects? Are there brain areas especially well suited for this prediction?

In Chapter 6, I look at a data set from a hospital, that contains labels for different kinds of noise in all channels. This is valuable, as it is a real world data set with a known distribution of samples, as opposed to the simulated data I use in Chapter 4. Unfortunately, this distribution cannot be used to verify the results from Chapter 4, since the noise labels may change too quickly. However, it offers the opportunity to take a more detailed look at the problem of identifying noise automatically, which is usually still performed manually by EEG experts. This problem can be severe, especially when the data is taken in hospitals, but it is usually much smaller when measurements are taken in a laboratory (like the smoker data set). The noise data set also offers another possibility to investigate the effects of the network structures that I adapted from pictures and applied them on EEG data.

Chapter 7 offers an overview of my research questions and their answers. The questions were motivated through many sections of this work, whenever a topic was explained that had touched my attention to dig a little deeper.

The detailed answers to the questions are given in the corresponding results in Chapters 4, 5, and 6. Therefore, a reader interested interested to build up his knowledge from scratch should read this work iteratively from front to back. If only interested in specific areas or research questions, I recommend to start with Chapter 7 to detect the research questions of most interest and then follow the references back to those sections where the questions were initially motivated and to the sections that hold the detailed answers.

Chapter 2

Background

In this chapter I explain all the background information necessary to understand the rest of this work. Since the source of the data I use is electroencephalography (EEG), I start explaining what it is, how it is measured, and which special challenges this data source entails. One of the challenges is noise in the data, thus I demonstrate several methods that can be applied during preprocessing in order to cope with different kinds of noise. The preprocessed data can be used in different machine learning models, from which I described the main ideas in Section 2.3. This prepares the following section on neural networks, which is described in more detail since some of my research questions focus on these models. During my processing, I split the EEG signal into several patches and perform predictions on those patches, which need to be aggregated into one prediction of the original signal. Before I investigate the performance of different aggregation methods, I explain them first in Section 2.5.

2.1 EEG

2.1.1 Measuring EEG

"Niedermeyer's Electroencephalography" [104] describes many works on the principles, the measurement and the evaluation of EEG signals. In the following I only give a short overview. EEG detects changes in neural activity by measuring potential differences on the scalp over time. Thus, it is non-invasive, and compared to other techniques very cost-effective. The data are measured at various electrodes relative to a reference electrode, and the electrodes cover the head in a way (see Figure 2.3) to optimally pick up neural signals, presumably generated from local field potentials [95]. Most clinical setups use 20 electrodes, but for research applications with the demand of a higher spatial resolution, so called *high-density arrays* with up to 256 electrodes can be used.

The temporal resolution is quite high, being on the order of milliseconds, whereas the spatial resolution is rather sparse, with electrodes being sparsely distributed over the scalp. Although EEG presents a great opportunity to

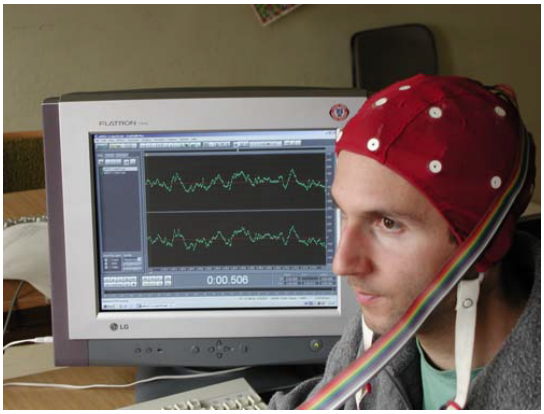


FIGURE 2.1: EEG Cap (Source: [144])

record activity, EEG also has drawbacks in that the electrodes pick up any source of electrical activity, whether neural or external noise, and therefore the signal is not as pure as one might hope.

In order to achieve measurements as clean as possible, a good connection between the scalp and the electrode should be given. In their book from 1982, Reilly et al. [121] recommend to use electrode jelly with 5-10% NaCl. They further suggest to prepare the skin by scraping off the most superficial horny layer. As this procedure is time consuming and aggravating, Taheri [143] developed the first *dry* electrode in 1994, which has become very popular.

However, this leads to the first artifact visible in EEG signal, namely loose electrodes (visualizations of many artifacts are given later in Section 2.2). But the EEG measures also other signals that are not generated by the brain. I start with the technical artifacts: Most important is alternating current, from power plants commonly at a frequency of 50 or 60 Hz. But there are also electromagnetic artifacts like inductive coupling. These often caused by the wires that connect the cap with the EEG device that are so close to each other that a change in current of one wire generates a voltage in the other wire by electromagnetic induction.

There are also biological artifacts: Sweating changes the conductivity of the skin, which causes slow shift movements in the signal. Muscle movements are very frequent: Bigger muscles show strong artifacts mostly in several channels, but for some muscles, characteristics are well known. For example heart beat can be identified by sharp spikes that occur regularly with each heart beat at a typical rate between 50 and 100 beats per minute. Further artifacts are created by muscles that are not as big, but much closer than the heart, for example chewing artifacts or eye movement.

2.1.2 Montages

EEG measures voltage fluctuations between pairs of electrodes. Those electrodes are located on the scalp of the tested subject. There are different stan-

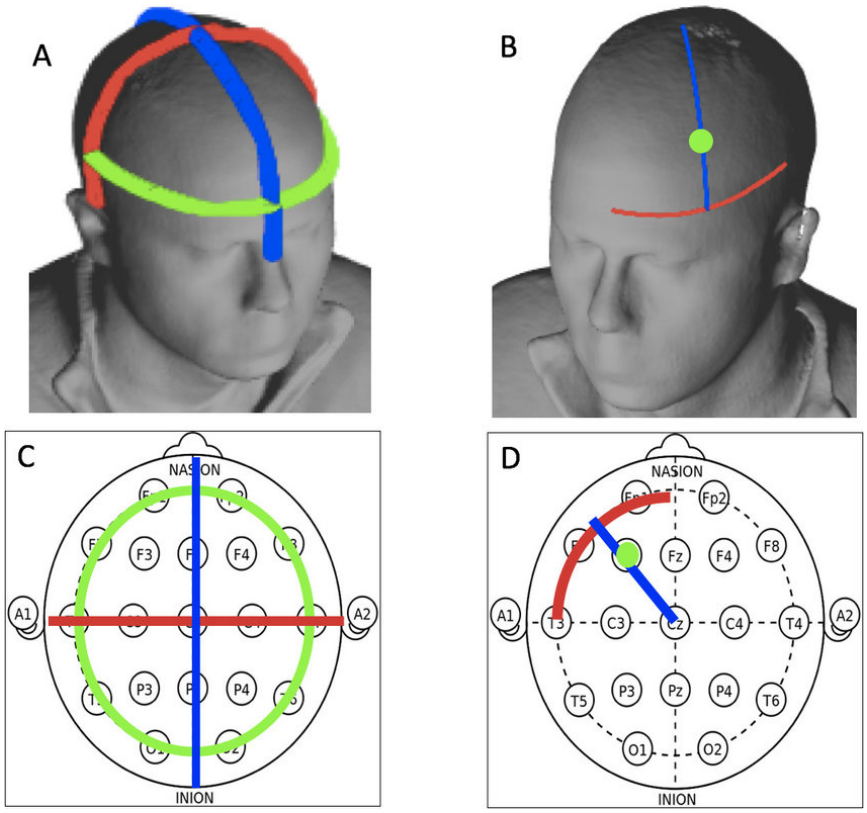


FIGURE 2.2: Electrode locations on the human head (Source: [98])

dards for the number and the positions of the electrodes. Most commonly used is the 10-20 system. The numbers *10* and *20* refer to the percentage of the distance between the two ends of the skull. From front to back this is defined as 100% as the distance between Nasion and Inion shown in Figure 2.2. From Nasion to Inion the distances start with 10% and go on with steps of 20%, until at the end another 10%. The same is performed with the imaginary line between the two ears. Note that this scheme generates 25 (10%, 30%, 50%, 70% and 90% along both of the head's axes) locations of which 19 (plus additional reference electrodes at the ears) are recommended for standard use [77]. I assume that one constraint is also the goal to have the locations set in roughly similar distances such that they cover similar areas on the scalp. Current EEG devices can use many more locations but most keep these 19 as basis. For example the 10-10 system extends the system by adding locations in the middle between the locations of the 10-20 system. Figure 2.3 shows the names and their locations on a schematic view from the top of a human head. Channel names in black are used in the 10-20 system. In addition to these locations, I marked all locations of electrodes I use in my later experiments in Chapter 5. Locations from the 10-10 system are labeled in purple, channel Oz (turquois) is one of the channels at the location of the 10-20 system, which is not one of the 19 commonly used channels, and finally there are the three channels O9, Iz, and O10 at the back of the head at 100%.

Some confusion might arise from the fact that since EEG was developed, some electrode locations were renamed and can have one of two names in

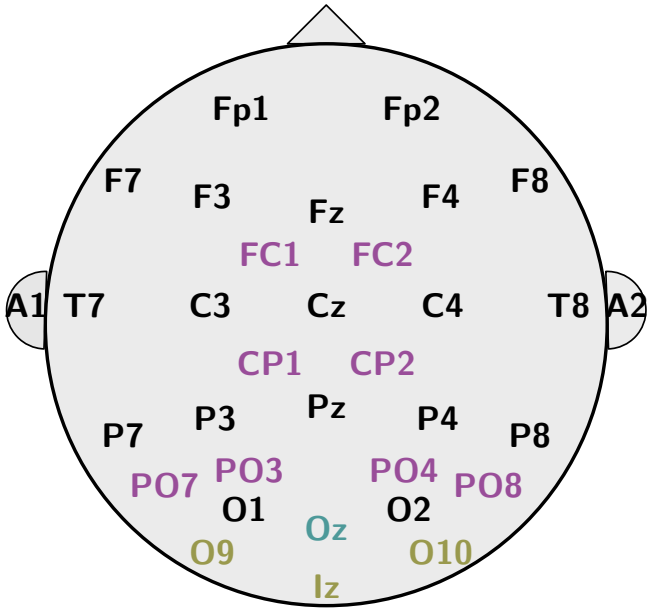


FIGURE 2.3: Schematic view of the channel locations on the human head

the literature: $T3 \leftrightarrow T7, T4 \leftrightarrow T8, T5 \leftrightarrow P7, T6 \leftrightarrow P8$. Note that in this work, I always use the same names T7, T8, P7, P8 for a better consistency while the names in the original literature may differ.

The American Clinical Neurophysiology Society offers a guideline for standard montages proposed to be used in clinical EEG [1]. EEG measures voltage fluctuations between pairs of electrodes. These pairs of electrodes are called *channels*. "Montages are logical and orderly arrangements of channels" ... "that display EEG activity over the entire scalp, allow comparison of activity on the two sides of the brain (lateralization), and aid in localization of recorded activity to a specific brain region." [1]. So a montage answers two questions: First, which electrodes are paired and second how they are ordered. There is a great diversity of montages used in practice. From the data analysis view, it is of smaller importance which montage is used, as long as the *same* montage is used all the time. Unfortunately, this is often not the case.

There are two groups of montages: *referential* and *bipolar*. In referential montages, all channels are computed by the difference of one electrode location and a fixed reference. This reference can be a fixed location, like the left ear (A1) or the right ear (A2), but it can also be a reference computed from multiple locations. When I speak of a channel and refer only to one electrode, this means that a referential montage is used and I omit this reference for better readability. For example the data used in Chapter 5 uses the right ear as reference electrode. When I speak for example of channel Pz, this means the potential difference of the electrodes at Pz and A2. Using a fixed reference electrode that contains strong noise, has a severe impact on the entire measurement, since all recorded channels will be affected. More robust is the *linked ear* reference, which uses the mean of both ear channels as reference. Another possibility is the *average reference*, which typically uses the average

of all channels as reference. Sometimes there are variations, that exclude such channels (for example Fp1 and Fp2, which contain muscle artifacts due to the muscle structure at the forehead) that are known to contain much noise from the average reference.

Montages are called *bipolar* if they use varying electrode locations to build their channels. It is common to use neighboring locations for bipolar montages and to use electrode pairs running in straight lines, best with equal distances between electrodes. Thus, typical bipolar montages are: longitudinal bipolar and transverse bipolar, which use straight lines of electrodes between the front and the back of the head or between the left and the right side of the head, respectively. An example that uses both is the transverse central parietal (TCP) montage system, depicted in Figure 6.9 (p. 163).

Both groups, referential and bipolar montages have advantages and disadvantages: Noise at specific electrodes can be detected easily by bipolar methods, as the noise is only present in those channels that use the noisy electrode. If a noisy electrode is used as part of a mean reference electrode, then this noise becomes more difficult to detect and to filter, because then the noise is also in the reference signal. For example, in a bipolar montage, it is easy to detect local changes, but difficult to detect synchronization between distant channels. Therefore, the American Clinical Neurophysiology Society [1] recommends "that both bipolar and referential montages be used for clinical interpretation".

Allowing different frequencies and montages gives practitioners more options, but makes an analysis more difficult. To maximize comparability between measurements it would be optimal to use a fully automatized measurement process, best with an individual cap for each individual that copes with the slight differences of the shapes of human heads. Until this is the case, one problem for analysts remains that there are systematical differences between EEG measurements.

2.1.3 Frequency Bands

So far, I discussed the *spatial* aspect of EEG recording. In the following, I set the focus on the *temporal* part. As EEG is a technique, that has been used as a tool for research and for diagnostics for more than 50 years, several insights have been gained. Achermann [2] found different dominant frequencies within the EEG signal during different states of sleep. Thus, a frequency spectrum, with their related mental states was found and is shown in Figure 2.4.

Note that the borders of the frequency bands differ depending on the considered literature. One advantage of Figure 2.4 is the visualization of the signal, but the gaps between the shown frequency bands are a drawback. However, insights on which mental state is associated with a given frequency band is

Frequency Band Name	Frequency Bandwidth	State Associated with Bandwidth	Example of Filtered Bandwidth
Raw EEG	0–45 Hz	Awake	
Delta	0.5–3.5 Hz	Deep Sleep	
Theta	4–7.5 Hz	Drowsy	
Alpha	8–12 Hz	Relaxed	
Beta	13–35 Hz	Engaged	

FIGURE 2.4: Frequency bands separation (Source: [20, p. 87])

important as it can be used during the preprocessing phase. Unimportant frequency ranges can be filtered if noise is present within them.

2.2 Preprocessing

The idea of preprocessing is to prepare the data such that they are comparable and can easily be processed afterwards. This later processing may be manually, e.g., by using data visualization techniques or automatically, e.g., by a machine learning algorithm. It is especially important that all measurements are identically treated, because otherwise the preprocessing *masks* the real data and thus may even falsify the actual results. If measurements were treated differently algorithms may be able to detect this and would use this as additional information in their predictions.

In the following, I use examples from two real world data sets: the smoker data set[1] explained in Chapter 5 and the Temple University Hospital (TUH) artifact data set from Chapter 6.

[1] For the smoker data set, the pre-processing was performed by Cedrik Pätz and myself with the help of Sarah Donohue. Sarah did not only teach us how to detect heart beat and eye movement, but also went through those data sets with us, where the decisions were difficult. I want to thank both of them for their friendly and kind collaboration.

2.2.1 Re-Referencing

The first necessary step to make measurements comparable is to make sure that they use the same montage. For the smokers, all measurements are taken with the same device and with the same montage, so there is nothing to do. This is different for the artifact data, which are given in three different montages. Therefore they must be converted to the same format, such that all measurements can be treated the same. This process is called re-referencing.

As it is important that all measurements use *the same* references, one might assume that it is of smaller importance which one is chosen — so one would probably chose the one which is most frequent to minimize the effort of re-referencing. Unfortunately, this assumption is wrong: For the EEG seizure corpus (another set of EEG data provided by the Temple University Hospital), Lopez found that the applied montage, and correspondingly the used reference significantly influences the performance [92]. Lopez compared two different montages: The Linked Ear (LE) and Average Reference (AR), which are also used in the TUH EEG Epilepsy Corpus. For seizures, the two montages are similarly frequent with 45% of the corpus. When trained and evaluated only on AR data, the model achieved about 60% accuracy, trained and evaluated on LE, more than 75%. When both data sets were combined, the resulting accuracy was mediocre. Their training sets contained data from 44 EEG records from each class and 10 records per class for evaluation. Unfortunately, the authors do not mention cross validation, which would indicate that their results may stem from a lucky or unlucky choice of evaluation data. However, other studies, like the work of Chella et al. [21] also show that the choice of the reference electrode has an impact on the outcome. Therefore, they recommend to perform a re-referencing of the channels.

As re-referencing is an active field of research [22], which is not on the scope of this work, I chose *not* to perform experiments on different montages. As the smoker data set in Chapter 5, the same montage (and even the same device) was used for all measurements, we kept this montage. The data from Chapter 6 are given in three different montage systems and are converted to the TCP montage system, as recommended by the Temple University Hospital.

2.2.2 Resampling and Fourier Transformation

Once data are in a format such that all measurements use the same channels, they are comparable w.r.t. the location on the scalp. But it is not given yet that they are also comparable in the time domain. This means some measurements are taken with higher, others with lower frequency. For the data set in Chapter 5, the measurements are all given using a frequency of 508Hz, so there is nothing to do. However, the Temple University Hospital artifact data set (Chapter 6) data use 250 Hz, 256 Hz, 480 Hz or 500 Hz as sampling rate. To make them temporally comparable they need to be *resampled* with

the same frequency. As most measurements use 250 Hz, it is the least effort to transform the others to 250 Hz as well.

The resampling process can be done in different ways. Obviously, just adding or removing values at the end of the signal and does not change the frequency, but only synchronizes the numbers of measurement points.

The common approach is to first transform the data from the time domain, where time points are mapped to values, into the frequency domain, where frequencies are mapped to values. Even though this representation was computed from samples, it actually describes a *continuous* signal in the time domain, that implicitly interpolates between the sample points. In this form, one can sample the signal at arbitrary points in the time domain. Doing so with a given new sampling rate results in the resampled signal.

As data is given by *discrete* measurements, I also apply the *Discrete* Fourier Transformation (DFT). Formally, this transforms a sequence of complex numbers X_k lossless into another sequence of complex numbers x_k with the same length by solving a linear equation system in the complex domain.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi n \cdot \frac{k}{N}}, k = 0, \ldots, N-1$$

Note that $e^{i2\pi/N}$ is the primitive Nth root of 1. Generally, solving an equation system with N variables and N demands $O(N^3)$ operations. But since this is applied on a commutative unitary ring, this can be used to speed up the computation. The *Cooley-Tukey algorithm* [27] recursively decomposes the computation of a longer series with even length N, into two series of the half length. More generally, the recursive algorithm by Schönhage and Strassen [129] can be used if N is not prime $N = N_1 \cdot N_2$ by computing N_1 smaller transformations of length N_2. These algorithms work most efficiently when the length of the given time series is a power of 2. Then it reduces the runtime to $O(N \cdot \log N)$. Note that some algorithms use further special properties of the data to improve the computation time in special cases, e.g., when the given time series consists not of complex but only of real values.

After the formula from above was used to create the equation system and solve it to compute the x_n, it can be used again for the resampling by computing $X_{k'}$ with $k' = 0, \ldots, M-1$ for the new sampling frequency M. Note that this resampling procedure is lossy, which means that there is information lost during the transformation. An example of original and resampled data is given in Figure 2.5.

Note that the signals can be close to each other, like between 2 and 7, but they can also differ. There can be differences between the original and resampled data, although the original data changes only marginally. This can be seen for x between 0 and 2, and for the curves at $x \approx 9$.

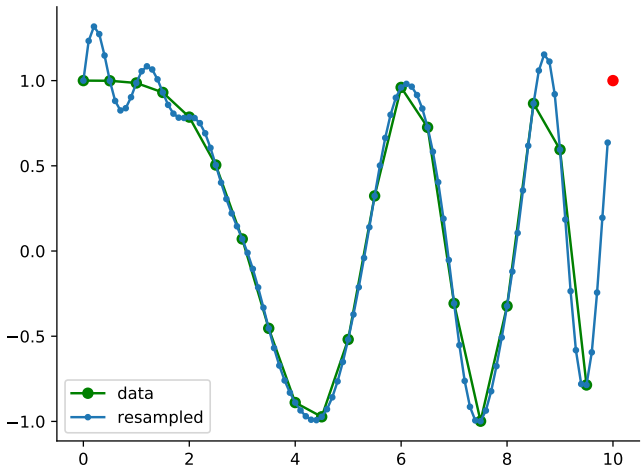


FIGURE 2.5: Example for original and resampled data (Source: [148])

When the data is resampled and back in the time domain, it can be passed on to a machine learning algorithm, as in each measurement spatial and temporal representations coincide. Note that it is also possible to use the data in the frequency domain as input for an algorithm. This representation is less intuitive for humans, but may generate useful features for a machine learning algorithm. For example, Roy [123] uses this space, and applies additional feature detection methods (see Section 6.2).

2.2.3 Detecting Noise

Noise can stem from different sources. Once the source of noise is known, the shape and occurrence can be understood and found by experts. Chapter 6 focuses on finding noisy parts of the signal *automatically* and classifying them correctly. But before I explain techniques to remove noise, I first explain how noise looks in real data. Eight samples of noise in EEG signals stemming from the TUH artifact data set are given in Figure 2.6. More details on the data set can be found in Chapter 6.

Before I describe different kinds of noise, I first need to explain how these plots are to be read: In many cases, a noise signal is so strong that it completely disguises the signal that actually stems from the brain. This makes visualizing the channels especially difficult. In order to compare channels of one plot easily, they need to use the same scaling. For a different plot, the scaling is fixed, but may be set differently. So in case of strong noise in one channel, this channel defines the scaling for the entire plot, but it does not affect other plots.

In Figure 2.6 I added a blue bar at the right side of each plot that indicates the scaling factor. A bigger bar shows that the y-axis was scaled down by a bigger factor. The biggest bar is shown on the top right plot with a factor

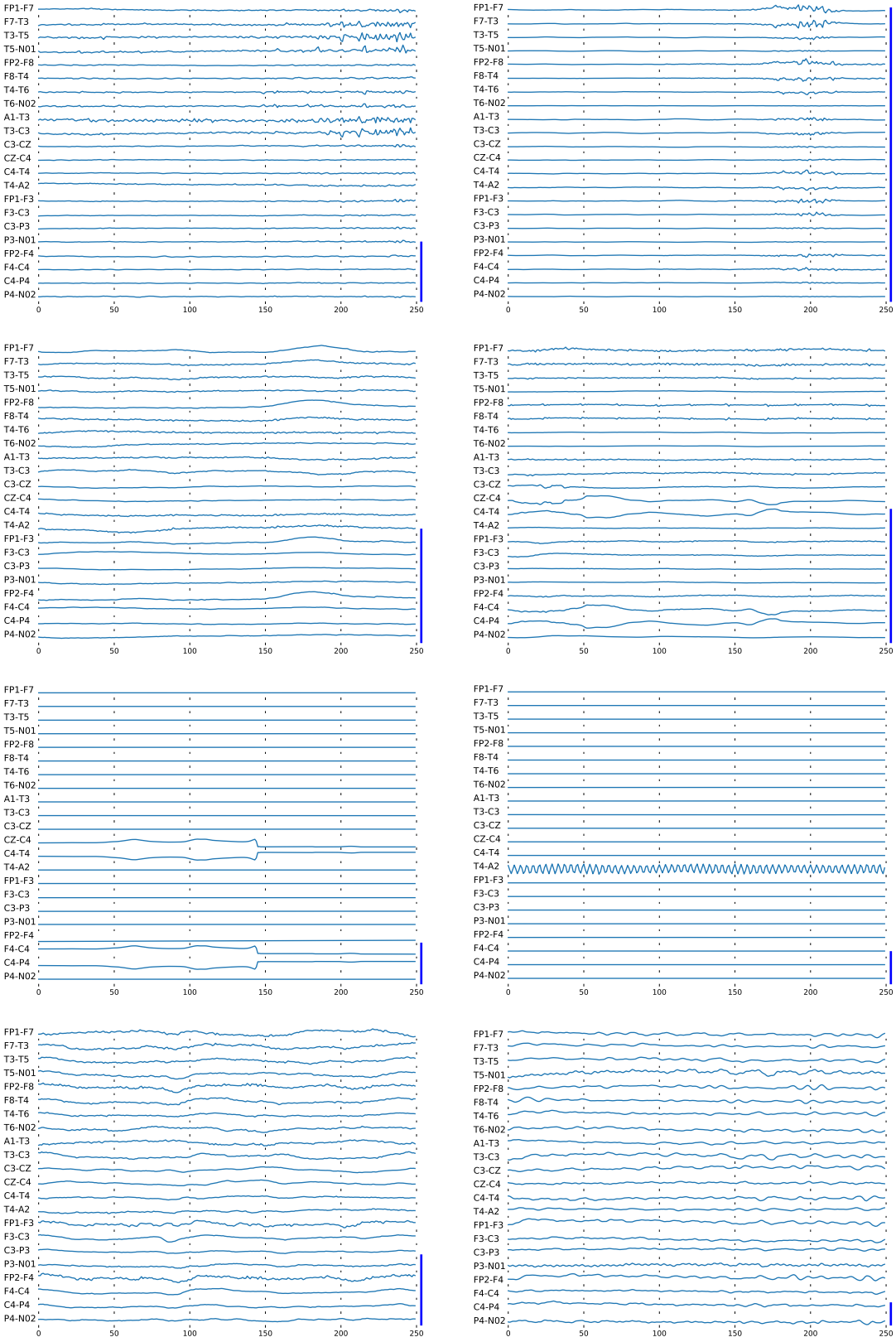


FIGURE 2.6: Examples for noise in EEG signals

of 460; the smallest one is on the bottom right with a factor of 36. The high factor of the top right plot shows how strongly the EEG can be influenced by chewing. The chewing artifact is best visible at an x-value of 200. As one plot shows one second of EEG signal measured at 250 Hz this means that the x-value of 200 corresponds to 0.8 seconds. Because of the strong scaling, there is hardly any visible signal, besides noise.

Muscle movement can be seen in the upper left plot by sharp, high frequent spikes. The effect starts at about 0.7 seconds and increases over time. Typical forms of low frequency noise is depicted in the second row. On the left, there is eye movement best visible between 150 and 220 which can be recognized by a low frequency wave of high amplitude, as indicated by the relatively large scaling bar. Another indicator for eye movement is that it is visible in the frontal electrodes Fp1 or Fp2. The right plot shows a loose electrode at electrode C4 that causes high amplitude low frequency waves. The last four plots were all labeled as clean signal. They have a small scaling, which seems to be a good indicator. From my point of view, row three is *not* a clean signal but shows an error of the measurement device. On the right plot electrode A2 shows a dominant 60 Hz frequency so I reason that this is caused by the power line frequency typical for the United States. As the scaling of the plot is small, but the other signals seem to be constant for the whole time, I conclude that the device is not working properly. The left plot has similar properties, but also seems *not* to be clean signal to me: The scaling is small and straight lines are visible in most of the channels. Only electrode C4 shows a visible pattern for which I can only guess which kind of noise it is. Finally, the fourth row shows plots that look like brain signal. But when investigating them thoroughly, one can also find possible noise here, too. For example electrode N01 in the right plot contains a 60 Hz frequency signal over the entire second.

The following three findings occur to me: First, in some snippets, the brain signal is completely covered by noise, in others only partially. So in fact the distinction between noise and brain signal in practice, is not sharp but rather fluent and changes over the time of the measurement. Second, experts also make mistakes. The ground truth may be wrong — as it seems like the two samples in the third row show no brain signal, but some kind of dominant noise. Third, the quality of a model to distinguish noise is limited by the given training data. Therefore it is mandatory to perform a pre-processing which ensures that those easily detectable kinds of noise are filtered out when the training of the model starts. Unfortunately, this point is hard to accomplish in a scientific environment that focuses strongly on new methods of machine learning and less on simple and robust techniques for data pre-processing.

However, I must emphasize the importance of the labeling work of experts. This is no task that can be done quick and dirty, but it builds the foundation of all later results. Thus, although I am neither a neuro-scientist nor an EEG expert, I strongly disagree with the expert, who labeled the third row of Figure 2.6 as clean signal. At the end of the day, giving *incorrect* labels can ruin

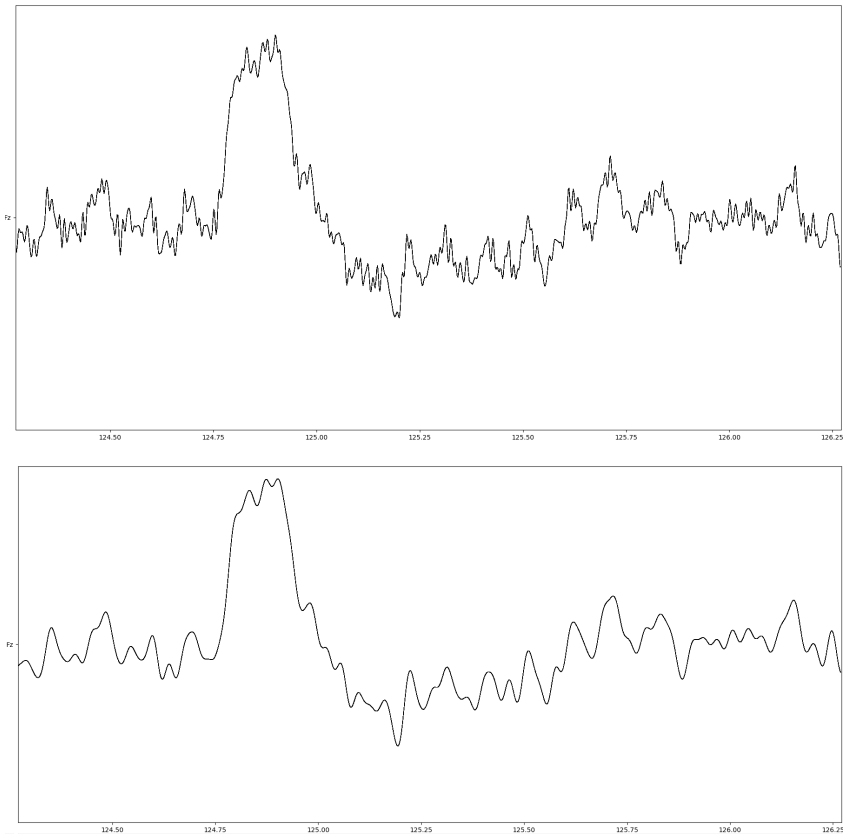


FIGURE 2.7: Signal example before and after band pass filtering (Source: [112, p.31])

the effort of anyone, who trusts in the correctness of the labels and uses them e.g. as input for a machine learning algorithm.

2.2.4 Removing Noise

In Chapter 6, the task is to *detect* noise within the data, and therefore I preserve the data as given. This is different for the smoker data set in Chapter 5. Thus, in the following, I first give general explanations and then use the latter data set as example to describe techniques to remove noise.

I am interested in only the brain signal. Signals from other sources are noise per definition. This noise can have several causes: muscle-related activity such as respiration, heartbeat, and eye movements. Sweat can impede the connection between the scalp and the electrode often adding a slow-drift to the signal. Electrical interferences at 50 Hz stemming from the alternating current of the power supply in Germany, where the measurements were done, is also frequently visible.

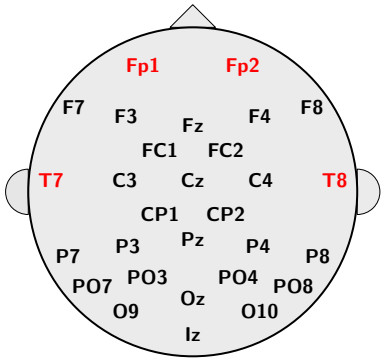


FIGURE 2.8: Overview of the electrode positions

Linear Filtering

Some kinds of noise only show up in specific frequency ranges. Knowing there is no or only little relevant brain signal within these ranges, one can apply a *linear filter* [136]. As the measurements here are taken at the resting state of awake people, one would expect the alpha channel, between 8 Hz and 12 Hz (see Section 2.1.3) to contain most of the signal. Thus one can apply a linear filter.

We decided to use a low-pass filter at 30 Hz, which lets all of the signal with a frequency lower than 30 Hz pass but filters the rest, and a high-pass filter at 0.5 Hz, which analogously keeps all frequencies above 0.5 Hz. As shown in the example in Figure 2.7, this procedure removed high and low frequency noise, including power line interference, some muscle artifacts, slow-drift related movements, respiration and sweat artifacts. At the same time it kept the frequency bands expected to be most relevant.

Removing Noisy Channels Completely

Removing physiological artifacts is more difficult, because they occur in the same frequency range as brain signal — linear filters do not help. One could for example cut those parts of the signal, where noise is visible, but this would induce new artifacts at the location where the cut is performed, or generate multiple parts. One can also remove noisy channels completely, but this means that the channels are not only removed from one measurement, but from *all* of them, in order to guarantee that all samples show the same data. Assuming that noise occurs rarely, this procedure would also remove much of the clean brain signal from many subjects. So this method is applied only, if all other methods to cope with noisy channels have failed. For several measurements, channels Fp1 and Fp2 showed very strong eye artifacts and channels T7 and T8 showed muscle artifacts that could not be removed. To prevent these artifacts from biasing the results, these channels were excluded from all subjects in the subsequent analysis to guarantee that the model cannot use noise as helpful information in its prediction. In Figure 2.8 these channels are marked in red.

Data Selection

Of course, it is possible to remove an entire measurement. But, as the number of measurements is mostly very small, as is usually the case with medical data, this is only the last solution, if nothing else works. Also, if not properly argued, it may be assumed that the authors intentionally removed those measurements that reduce their prediction quality in order to improve their results. Colloquially speaking, this process is called *cherry picking*.

For the smoker data set three participants had to be excluded completely: One had fallen asleep during the recording, and two more were rejected, as it was impossible to remove the artifacts without removing most of the signal as well. For the analysis, data from a final set of 27 smokers and 9 non-smokers remain, each with 25 channels using the right ear as reference electrode. Note that we did *not* perform cherry picking but removed the subjects after the preprocessing phase and *before* any model was applied.

Independent Component Analysis

A less radical possibility to remove physiological artifacts is to apply an Independent Component Analysis (ICA). In this process, a multivariate signal is divided into additive subcomponents, for which it is assumed that the components are statistically independent. In order to do so, first a principal component analysis is applied which is used to whiten the data set. This means that after the whitening, the resulting covariance matrix is the identity matrix. Then the algorithm determines an affine transformation (in my case a rotation) such that an independence criterium for the new space dimensions is met. One possibility is to maximize the difference to a gaussian distribution, another is to use the negentropy. For more details I recommend the work of Hyvarinen [71].

Independent components from the signal can be computed automatically, taking a few minutes for one measurement of the smoker data. The motivation to apply this approach to remove noise is that the statistical independence of the components splits the data, such that there are some components that show strong noise and only little brain signal, and others with little noise and strong brain signal. Thus, for each of these components it must be decided — best by an EEG expert — whether it is to be kept (accepted) or to be removed (rejected). The expert has several indicators that help him with the decision. The following three indicators were most helpful for the preprocessing of the smoker data:

First, correlations of the components and single channels can be used. Because the frontal electrodes Fp1 and Fp2 are known to show eye movements and eye blinks, high correlations with these channels indicate that this noise is also dominant in the component. Similarly, the channel of the left mastoid was often correlated to a component showing heart beat.

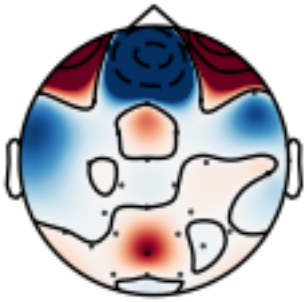


FIGURE 2.9: Example distribution topology (Source: [112, p.32])

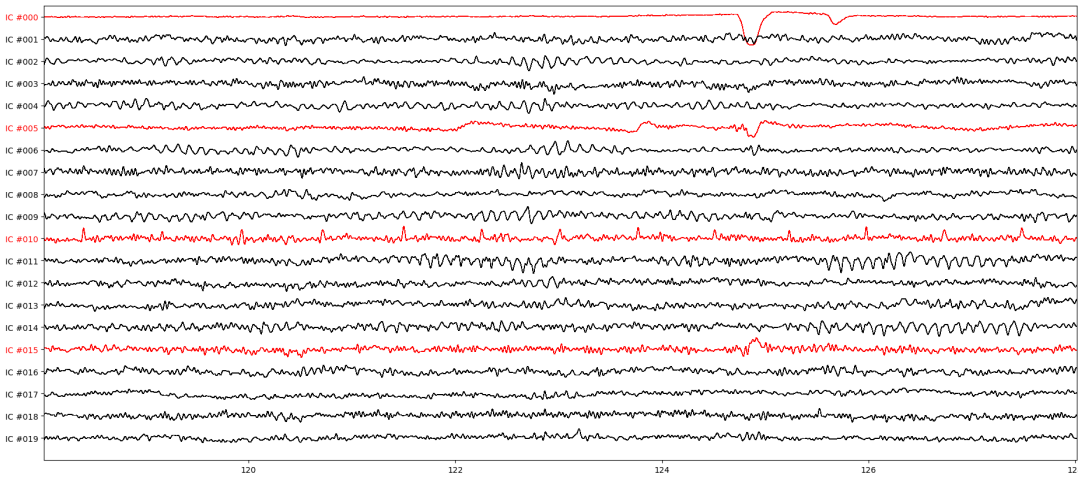


FIGURE 2.10: Component selection after calculating the ICA (Source: [112, p.33])

Second, for each component, the distribution topology can be drawn. This image summarizes the component's source channels. The example given in Figure 2.9 implies heavy eye blinks, indicated by the dark red areas in the frontal region. The stronger the color, the stronger the occurrence in the underlying component, red showing a positive and blue a negative factor.

Third, the generated components can be drawn as time series, just as the input signal. Characteristic signals like eye blinks or heart beat can often be detected easily. The example in Figure 2.10 shows 10 seconds of 19 components. The four components in red are to be rejected, the others are kept. Component 0 shows an eye blink, component 5 contains a strong drift before the eye blink. The rhythmic spikes in component occur with a rate of approximately 80 per second. This, and the typical wave form are clear signs of heart beat. Finally, component 15 seems to contain mostly clean signal. But this does not hold for the entire measurement, which the expert has to examine thoroughly. In parts that are not shown in the Figure, strong noise is visible and thus this component is removed as well.

As most of the components do not exclusively contain noise or clean signal, this is always a trade off. Especially when a signal of several minutes has to be checked, it is likely that some intervals of a channel contain noise, while other intervals are clean. The expert has to ponder which ones to choose.

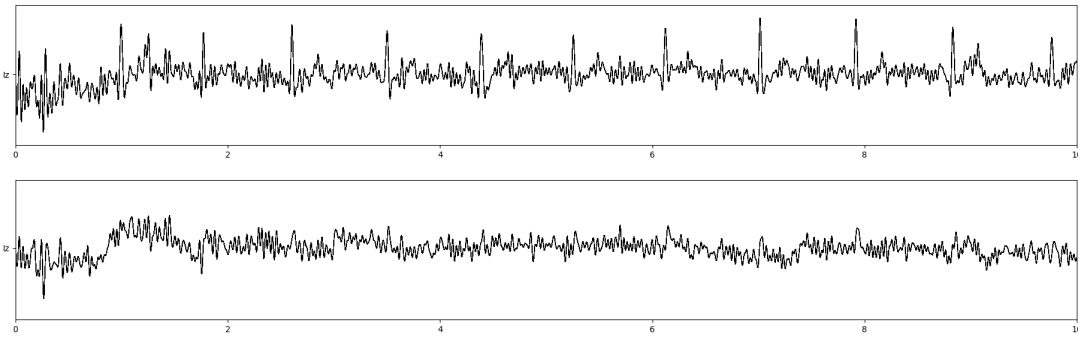


FIGURE 2.11: Signal with heartbeat before and after the ICA was applied (modified from: [112, p. 34])

Once the selection is made, the signal is projected back to the original space, keeping only the accepted components. The thereby created signals are then visually verified, to check whether the strongest noise has been removed and brain signal was kept. Figure 2.11 demonstrates the successful removal of the heart beats from channel Iz.

So applying the ICA on a data set is a semi-automatic process, which has to be performed for each measurement. The computation of the components is done automatically, but the rejection of them needs to be done manually. This process is best performed by an EEG expert, and as he needs to detect noise visually, this task can be tiring and very time consuming.

Running the ICA once can have three outcomes. First, there may have been a convergence problem during finding independent components. In such a case the process can be run again with another initialization. Second, the visual inspection shows that too many components were removed such that the resulting signal does not contain much visible brain signal anymore. In this case, one would restore the original signal and remove fewer components in the next run. Third, if after removing noisy components there is still visible noise, then the ICA can be run again. Since some components were already removed, a new run will find new independent components. For the sample data set, the selection of the components was conservative, as the primary goal was to keep as much brain signal as possible.

The generated signal should afterwards be visually verified to make sure that the artifacts were successfully removed. This was the case for the smoker data set, but the independent component analysis had also created high frequency noise. However, this noise could easily be removed by a linear filter that kept only the signal between 0.5 Hz and 30 Hz (as before).

Channel Interpolation

When noise dominates one or a few channels, they can be completely removed and replaced by interpolations created with the help of the signal of the surrounding channels. Perrin et al. [114] describe a process, in which they

use spherical splines for these interpolations. I considered this process to replace the frontal channels, if they were very noisy. The interpolated channels showed very little noise, but also hardly any visible brain signal. In order to give the algorithm no visible sign of preprocessing, I removed the noisy channels Fp1, Fp2, T7 and T8 from *all* measurements, instead of interpolating them for the noisy measurements.

2.2.5 Normalization

Machine learning algorithms normally work on plain numbers and omit the units, and also the units' scales. Although it makes a difference whether a unit is measured in meters or kilometers, as long as the proportions are kept, the outcome should not differ when the input is given in the one or the other unit or with one or the other scaling. Some algorithms, like decision trees, have the inherent feature to ignore the scaling — other algorithms like neural networks do not: I consider a network with only one layer, which is initialized (more about weight initialization in Section 2.4.8) with weights in a fixed way independent of the input variables, then for differently scaled variables the outcome of this layer differs, and thus does the output of the network.

To cope with this problem in general and for arbitrary algorithms, the input variables should be *normalized* to make sure that all input variables are in the same range and similarly distributed. This can be done by using the minimal and maximal values, but as this process is fragile when there are outliers. Therefore, the standard is to apply a *z-score* normalization. This is done by taking all (or as approximation a sufficiently large subset of) input data, subtract the mean and divide by their standard deviation, such that the resulting variables have a mean of 0 and a standard deviation of 1.

However, although this process is conducted in order to enable reproducibility, it may have negative effects on the predictions. When for example few strong outliers affect mean and variance, the normalization process impacts the values of all samples. Thus outliers should be filtered in advance. This becomes especially difficult in Chapter 6, as the task is there to detect noise and as such, outliers should be kept in the data set, but should be ignored, when computing the normalization values. When this is not done carefully, it can have confusing consequences, for which an example is given in Chapter 6.2.

For time series, the question is raised on which level the normalization should be performed. Should each session be normalized individually? Each channel in each session? Should the measurement device be considered?

For the artifact data set in Chapter 6, I decided to keep the processing pipeline unchanged to gain the best comparability possible. For the smoker data set (Chapter 5), all channels were normalized individually to have a mean of zero, but I kept all the variances. Different variances may be caused by the

brain and should therefore be considered in the training process, or they may be caused by noise in the measurement and should be filtered out. As one cannot be sure, I decided to keep them. The MNE framework [53] was used for the entire preprocessing. This brings me to my first research question:

Q1: *What is the effect of preprocessing on the prediction quality?*

2.3 Machine Learning

To understand complex relations in reality, humans have developed mathematical models. These models follow strict rules and thus allow predictions in the model space. This can be of practical use if their predictions can be transformed back and allow predictions or give insights on reality. If the models do not allow accurate predictions, one approach is to model the problem differently, possibly with another approach, or a different data representation. Simple (linear) mathematical models are taught at school: "If one man needs one day to dig a hole of size $1m \cdot 1m \cdot 1m$, how long does he need to dig a hole of size $3m \cdot 1m \cdot 1m$?" With three times the size the linear model predicts the bigger hole to take three times as long to dig. Although in reality the ground may be different for the bigger hole — which is implicitly assumed to be identical to the one of the small hole —, the result is a prediction that can be tested in reality. In this example a *linear* model is specified and training samples of sizes of holes and the respective durations are used to determine the optimal values of the model parameters such that the error on the training data is minimized. To verify whether the model works well, also on unseen data, it should be tested on unseen data afterwards. In the example, a fixed speed of digging is defined, which is the only model parameter. Note that the model is intentionally chosen to be as simple as possible for several reasons; it is easy to compute and easy to understand. This simplicity also limits the possibilities to adapt. For example it does not consider that for deeper holes it becomes more and more difficult to remove the rubble. To create a model which is able to respect this fact, the modeling process needs to be performed again to build a model with more parameters. Afterward, it needs to be trained on the given data to set the values of the parameters accordingly. For models of the same type, it can be shown that those with more parameters make at worst equal errors after successful training. Thus, there is a tendency to create more complex models, with more parameters, as their errors are normally smaller. The drawback is that each additional parameter also makes the model more difficult to understand, it needs more computations to train and it is more prone to overfitting (see Section 2.3.2).

So far I described the mathematical modeling of a problem. Machine learning adapts the model parameters with the help of given training data. Assuming a data set with n samples is given. Each of the samples consists of dependent variables x and independent variables y. The model uses the dependent variables to predict the independent variables. During the training

process model parameters are adapted in order to minimize the errors made when making a prediction.

In a classification setting, the values of y belong to a fixed set of classes or *categories* and are thus called *categorical*. When the predicted variable y is continuous, then the prediction task is called *regression*. For a wider explanation of the huge field of machine learning and artificial intelligence, I recommend the work of Russel and Norvig [125]. A more detailed overview on the data analysis process can be found in [11]. I focus on those parts of the process most relevant for EEG data: data preparation, modeling and evaluation.

In the following section I explain several aspects of machine learning relevant to this work in more detail. This builds the basis of the explanations of the methodology chosen for my later experiments. Then I present and explain a few machine learning algorithms. Since the main focus of my work lies on neural networks, I devote a separate section (Section 2.4) to them.

2.3.1 Modeling

During the modeling process, the mathematical model and its parameters are defined. Often, the literature defines the modeling process as follows [11]:

1. Selecting a model class or model architecture
2. Selecting the score function on which the model is to be optimized;
3. Applying an optimization algorithm that set the parameters such that they are optimized with respect to the score function
4. Validating the result

In the digging example from above, the number of parameters and their values are given. In a machine learning task, the model architecture needs to be initially chosen and afterward the parameters' values are determined with the help of the training data during the training process. This process starts with initializing the values. In an implementation, first memory for the needed variables is allocated and depending on the model the variables may already be set to default or random values[2]. Thus, in this state, the model may perform random predictions. For example a simple model could ignore the input and always predict the same value, or in case of classification, always the same class. The training process then *fits* the model's parameters to the training data. Implementations of some models demand that all training data is given at once, such that all computations can be performed directly. However, in times of *big data*, this approach is limiting, since even though the available memory is constantly increasing, it is beneficial to be able to *adapt* a model iteratively. This allows that the computer memory is smaller than

[2]Often, the model is what is produced by the four steps explained above. In contrast to the definition in the literature, I prefer to use the term 'model' already after step 1, when I would call it *unfit* or *raw* model in contrast to *trained* or *fit*, after the training.

the training data and allows for an adaptation even after a first training is completed. The training process of neural networks performs such iterative adaptations, which I explain in more detail in Section 2.4.

During the modeling and machine learning process, the model is optimized under the assumption that the given data allow proper predictions. The problem with this assumption is that it is unknown whether it is true or not on new data. Often, the reverse approach is applied: the model is trained on the data, and when it performs well the assumption is probably correct. If it does not perform well it cannot be decided whether it is impossible with the given data, or the model was badly chosen. When the assumption is false, then the model will adapt towards random noise and will therefore over-fit the training data. Interestingly, another effect can also occur: the models performs very good, but it did not learn what it was intended to learn. I describe a anecdote on how this can happen in the following.

Model Understanding

One of the most important aspects of modelling is to create a model in a way that can later be easily understood. I heard this example at a conference, but was unable to retrieve it and thus can give no citation:

A student was asked to create a classifier to automatically find out whether or not an animal was in a picture. As it is only a binary decision this seems like an easy task. But consider that there are thousands of different animals, and pictures could show them from an arbitrary angle. Maybe, the task is not as easy as initially thought.

With many pictures available online, the student was able to create his own database with pictures. The first classifiers trained on them were surprisingly successful, achieving more than 90% accuracy. This result seemed too good to be true and thus professor and student investigated which features the classifier had used. They found that the most important information was at the borders of the pictures, which is confusing as animals are normally seen in the center of a picture. But what can be seen at the borders? The camera's focus! Pictures with animals show them mostly at the center, and the focus lies on them, while the background is blurred. If the background is sharp, this indicates that the picture shows only landscape and thus contains no animal.

The speaker concluded the anecdote, saying that one should not just care about the prediction quality, but one should also always try to understand the created model. Otherwise there is the possibility that the model has found something, but not what was actually intended. When only aiming for a high score, one risks to miss important insights.

My personal conclusion was that in each training data set there is a bias, caused by the way the data is measured. In this example the bias stemmed from the fact that the student had created his data set using a picture search

engine. I guess he made a list of animals and looked up pictures for each entry to create the samples of the animal class. For the second class he probably googled for landscape images. This is plausible as googling for pictures without animals actually returns pictures *with* animals. One would probably search for another subject, just as landscape images. And landscape images probably have similar color histograms compared to pictures of wild animals. So at least from this point of view the choice is appropriate.

I agree with the speaker's conclusion and I am aware of the fact that I am not able to verify the correctness of my models myself, because I am neither an expert on neuro-science nor on EEG measurement. For the smoker data set, for which experts claimed it to be impossible to distinguish smokers from non-smokers by their EEG, I follow two approaches: First, I use a random permutation test to verify whether or not the trained models are able to create plausible predictions, while the same should be impossible when the labels are randomly shuffled. If predictions were (on average) better than guessing, although the labels were shuffled, this would indicate a severe problem in the model, the data or the processing. Otherwise, the test reassures that the results are not caused by chance but by an underlying systematic relation. Second, I create a visualization (Section 5.5) which shows the input signal and also the model's predictions for the given signal. This builds a tool for experts enabling them to visually validate the trained models.

Bootstrapping

The most costly part of taking EEG measurements is to find subjects, and prepare them and the EEG device for the measurement. Taking a *longer* measurement is mostly not a problem. Therefore it is typical that EEG data sets contain few but long measurements. In terms of data analysis this means that EEG data sets contain few samples with many dependent variables. This can become problematic, because this increases the chance that the model is fit to patterns that coincide only by chance and not by *actual* relations between the dependent and independent variables. Therefore, one important part of modeling should focus on this issue.

One approach is to use *bootstrapping* as a method to generate more training data from a given set of samples. There, each measurement is partitioned in time, to create many smaller samples (snippets). The downside of this approach is that the samples generated by this bootstrapping are not independent, as they are taken from the same measurement. This has consequences for the validation process which is presented later in this chapter. Another issue is that mathematical models typically use a fixed number of dependent variables as input and thus this number needs to be chosen. Fewer dependent variables allow for more snippets and vice versa. Which value yields the best-performing model needs to be found by trial and error. Note that an example of this process in explained in detail in Chapter 5.

2.3.2 Validation with Stratification

So far, I only stated that the models need to be validated on an independent set of samples in order to allow estimates on unseen data, but I omitted where these samples come from. The easiest way is to use only a subset of the available samples for training, and the remainder as an independent set for testing. This way the samples are split into two disjoint subsets, the training set to *fit* the model to the data, and the testing set to evaluate the quality of the model. In order to ensure valid results during the validation (or testing) process, one has to assure independence. Thus, samples taken from the same subject are not allowed to be contained in the training set and test set simultaneously. During testing, the model is only given the dependent variables, in order for the model to predict the independent variables. The real values of those independent variables are already part of the original data, so the models prediction quality can be evaluated, when compared to those preexisting values.

It is common to reuse data several times in independent tests in order to get a reliable estimate of the quality, for an unknown, unseen data set. A good overview of cross-validation procedures was written by Arlot [3]. For applications, one is interested in good predictions for new, unseen data. Thus, one must assure that both, the training set and the validation set share the same distribution with the unseen data so that all three sets are *identically distributed*.

An easy way to get probably independent distributions for training and validation data is to sample them independently from all available data without replacement. When trying this once, one might be unlucky, as the distributions might differ because of chance. In this case the prediction quality on the validation set differs from the actual quality on the new data. Nevertheless, this sampling can be performed more than once independently. Thus one can expect the resulting distributions to be independent on average. The more often one samples, the higher is the probability of the estimator being close to the actual quality. This process of repeatedly sampling training and validation data is called *cross validation*. When the number of available data samples is small, the variation of the samples is expected to be relatively high. Thus, one should repeat this process often in order to receive most reliable result. The drawback of many repetitions of this process is that one model needs to be trained and tested for each repetition. So the actual run time of training and testing a model needs to be multiplied with the number of repetitions. Note that this procedure does not evaluate a model, but rather a model building procedure on the given data. Thus, it is meant to allow a generalization: With data of this type, if a model (of this type) is built on it, one can expect the model to have, on average, this accuracy.

After the model is trained, one possibility is that the model successfully learned all training data by memorizing them. So, when tested on these data, it reproduces them without mistake. But when it has only learned those data and nothing else, then for all other data, the predictions are still random.

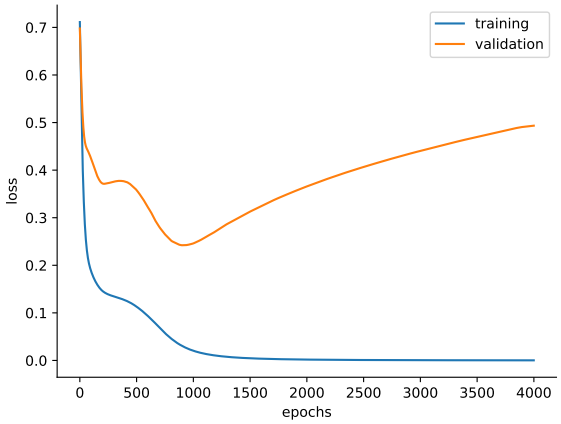


FIGURE 2.12: Example how model loss changes during training (Source: [18])

What is actually desired is a model that is able to *generalize*. To test successful generalization, one needs to perform tests on a data set that has not been used for training — the validation (or testing) data set.

An histogram of one model's performance during training on the training and the validation set is depicted in Figure 2.12. For more epochs of training, the loss (that represents the error and is explained in Section 2.4.4) reduces. Until epoch 900, the model is still *underfit*, which means that more training reduces the model loss. Note that the relevant value is the one computed on the validation set, which reaches its minimum at about epoch 900. From this point on the loss keeps decreasing on the training set, but increases on the validation set. This is the point during training, when the model starts to *overfit* to the training data. Before I describe the most widely used cross-validation methods, I first explain the problem of class imbalance, as it complicates the validation methods.

Class-Imbalance

As stated above, in a classification, it is recommended to have the same distributions in the training set, in the test set and in the overall data set. These distributions refer not only to the dependent but also to the independent variables (the classes). If the classes are not equally frequent in the overall set, this should also be respected by the splitting procedure. Sampling procedures respect class-imbalance by not sampling randomly, but by considering randomly from the sets of the classes. As the numbers of samples per class may be arbitrary, this may not be possible in a perfect way due to the integer numbers. However, it can be approximated by the next integer value. Procedures that respect imbalanced classes are also called *stratified*.

Shuffle-Split Cross-Validation

The simplest approach to create splits in a cross-validation performs a random shuffle and splits the so shuffled data into training and test data with respect to the split ratio. Typically values lie between 50% to 90%. At this point of the explanation, this method is just a validation, as it does not use use the data set several times. So repeating this method makes it a *cross*-validation. A nice property of this algorithm is that the number of repetitions is not fixed beforehand. For example when the results of a few runs show a high variance, one can simply perform additional runs. Because all runs are independent, results from old runs do not have to be computed again. Of course, there are practical limits on the number of iterations. It can happen that a split of a new repetition is by chance one that has been used before. However, the chances for this to happen are small. To be sure that the experiments are reproducible, each run should use a unique random seed value. The seed values for my experiments are given in `data/seeds.csv`. In practice, two ways of cross-validation are more widely used: folded- and Leave-K-Out-cross-validation.

Leave-K-Out Cross-Validation

The first famous form of cross-validation is the so called *Leave-K-Out*. K samples are left out for training, they are used for validation. The remaining $n - K$ samples build the training set. This is repeated for every possible subset of the given data with K samples. This procedure allows extensive testing, even for data sets of small size. However, the parameter K is typically 1 (then the process is also called "jackknife" in statistics), as the run time increases tremendously with increasing K, because all possible subsets of size K have to be considered. A Leave-1-Out even needs to train n different models. For $K = 2$ this number is already $(n^2 - n)/2$. The number of repetitions is given by the binomial coefficient $\binom{n}{K}$. So this method is clearly made for small values of n and even smaller values of K. Between two iterations, the training data differ at most at $2K$ samples. Therefore it is likely that the resulting models are also similar over the different runs as this process examines only a small area in the model space. Therefore it can happen that the resulting value is a bad estimate for the model performance on a new, unseen data set.

Folded Cross-Validation

One drawback of the shuffle-split approach — especially for big data sets — is that in each repetition all samples need to be considered. One approach to avoid this computational effort is to group samples into K disjoint subsets, the so called *folds*. Each folds distribution should resemble the distribution of the whole data set as much as possible. Instead of sampling from the original data, one can randomly sample folds to create training and test set. As all folds resemble the same distribution, unions of folds resemble it as well.

In a K-Fold Cross-Validation, there is a total of K folds of which the union of $r < K$ folds builds the test set and the samples from the remaining folds are used as training set. It is typical to choose $r = 1$ and use the same sampling as for Leave-K-Out and receive $\binom{K}{r}$ possibilities to sample folds. Common values of K are 5 to 10 in practice. For a fixed split ratio of e.g., 80% one can choose $K = 10$ and $r = 2$ or $K = 5$ and $r = 1$ depending on the desired variation in the splits.

Folds are typically generated by random sampling. Therefore, the chances are high to have similar distributions in the folds and the overall data. Still, as folds are only created once, it is possible to be unlucky. As randomization causes this problem, repetition can solve it: One can repeat the entire process, create different folds several times, use them to perform cross-validations and at the end aggregate all these results. This way, one accepts longer run times in order to minimize the estimation error. The most extreme case is to set $K = n$ and $r = 1$, creating one fold for each sample, which is equivalent to *Leave-One-Out Cross-Validation*.

Grouped Validation

When working with measurements from subjects, the problem of independence of the data occurs again. It is not viable to consider two measurements of the same person to be independent. In fact the assumption that we can learn from person-specific data for any prediction model already implies that the results should differ across subjects. It also implies that for the same subject, the results should be similar, and this means that they are not independent. This is also true for several snippets created by bootstrapping from one measurement. Therefore, validation methods should account for these dependencies. *Grouped* validation methods perform similar methods as the normal ones, but they also guarantee that all samples within one group are used for the same purpose. All are used for training or all are used for testing.

So how can the validation algorithms be modified to guarantee this property? They apply the split not on the samples directly, but on the groups. Grouping can be applied to most of the cross-validation methods explained earlier. Although this sounds simple, there are some practical implications which make the validation more difficult, especially when there are only few groups or the number of samples vary strongly. For example in a folded cross-validation all samples of one group need to be in the same fold. Further, the variation within the validation sets is pretty high. Therefore it is less probable (compared to non-grouped) that validation sets represent the distribution of an unseen data set very well. So the outcomes are less reliable.

Grouped and Stratified Validation

A general solution would allow for arbitrary groups of samples and allow for stratification to achieve (perfectly) balanced classes within training and

test set. The problem is that the given groups may induce constraints that are impossible to satisfy: For example if all samples of one class belong to the same group, they can all be used for training, or all be used for testing — in both cases the class distributions of training and test set would differ. In such a case, in order to minimize differences between training and test set, it would probably be best to exclude samples of this class completely.

I describe the general problem formally. Given are a split ratio s, n classes $c_1, \ldots, c_n$, with the vector $\vec{\#c}$ of samples per class and m groups of samples $g_1, \ldots, g_m$ and for each group a vector r is given that describes the ratio of the classes contained in the group. A solution is a vector of binary values $(b_1, \ldots, b_m)$ defining whether the corresponding group is used for training or for testing. A perfect solution makes sure that each class is contained in the training set with exactly $s \cdot \#c_j$ samples:

$$\sum_{i=1}^{m} b_i \cdot \vec{r}(g_i) = s \cdot \vec{\#c}$$

This can be interpreted as a multi-dimensional knapsack problem [46], which is a combinatorical problem known to be NP-hard, already for one dimension. For the multi-dimensional case, it is shown that there is not even an efficient polynomial time approximation scheme for $d = 2$ (meaning 2 classes, in my case) unless $P = NP$ [85]. So it is clear why there is no general implementation given in the common machine learning frameworks: In the general case, even approximation heuristics like genetic algorithms, or ant colony systems [110] take too much time to compute.

But, knowing that a *general* solution is hard to obtain does not necessarily mean that this holds for each problem instance. In my problem, Chapter 5, the groups are similar: each individual person must be considered a *group*. Thus, some groups contain only samples of one class (non-smoker), and others contain samples from exactly two classes (craving and non-craving). Further all measurements have the same length, which makes the 'groups' of smokers as well as the 'groups' of non-smokers interchangeable. This simplifies the problem and allows an efficient computation, which is described in Section 5.2.

2.3.3 Model Evaluation using Class-Balanced Accuracy

Imbalanced classes do not only complicate the situation when creating the training and testing split, but also when evaluating the model quality. The most detailed version of how to evaluate a classification model, is by considering the confusion matrix. This matrix gives an overview that shows for each pair of classes (predicted class and actual class) how many samples were *confused* by the model. Thus, one gets an overview not only on the absolute numbers of samples, but also on which confusions occur.

This is a great tool to compare several models that were applied on the same data set. However, it has the drawback that this comparison becomes difficult when the data sets differ. This can happen for example for different splits of the same data set, but also when comparing models working on different data sets. One difficulty can be that the class distributions are different, e.g., one class occurs more frequently in one data set than the second. To enable comparability over different data sets, one approach is to sum up all correctly classified samples and divide it by the total number of samples to create the *accuracy* score. In terms of the confusion matrix this means to add the values on the main diagonal and divide by the sum of all values. Note that dividing by the sum can be seen as a normalization, which improves comparability, as it results in values between 0 (all predictions are wrong) and 1 (all predictions are correct), independent of the number of samples in the data set. When the data set is strongly imbalanced, models that predict the *majority* class can get a higher score more easily. An extreme model would *always* predict the majority class and would yield an accuracy of the relative frequency of the majority class. Since the thereby computed accuracy score weighs each class with its frequency, it is also called *class weighted accuracy*.

One approach weight all classes equally in the evaluation is to perform a *balancing* across the classes to compute the *class-balanced accuracy score*. Note that it is equivalent to the class-balanced F_1-Score [130] with micro-averaging. Let n_{ij} be the number of samples predicted to be class i but actually belonging to class j and

$$n_{.j} = \sum_{k=1}^{c} n_{kj}$$

the sum of samples belonging to class j. Then the balanced accuracy is computed as

$$acc_{balanced} = \frac{1}{c} \cdot \sum_{k=1}^{c} \frac{n_{kk}}{n_{.k}}$$

Simple Models

For the balanced accuracy score, any classifier that always predicts the same class (independent of the number of occurrences) yields a class-balanced accuracy of $\frac{1}{c}$. In my experiments, I always compute the class-balanced accuracy score. For better readability, I omit the term *class-balanced*. When considering the accuracy score which does *not* consider balancing, I refer to it as *unbalanced* accuracy.

Another very simple model performs random predictions. In the most trivial form it does not respect the class balance but gives every class the chance $\frac{1}{c}$. When it does respect class-imbalance, the chances are $\frac{1}{n_{.j}}$ for each class j. The mean accuracy values for majority prediction and random guessing are obviously identical. For limited number of iterations, there are however vari-

ations for random guessing, which do not occur for a model that predicts always the majority class.

A model, which does not only return a class label, but class probabilities and is optimized to make errors as small as possible may converge to an input-independent state, where it returns always the same output vector. If the classes are balanced or the optimization process respects class-balancing, then the resulting output vector is $\vec{v} = (\frac{1}{c}, \ldots, \frac{1}{c})$. Otherwise one can expect the vector $\vec{v}$ being identical to the vector of relative class frequencies $\vec{v} = (\frac{n_{.1}}{n_{..}}, \ldots, \frac{n_{.c}}{n_{..}})$, with $n_{..} = \sum_{k=1}^{c} n_{.k}$ the sum of all samples.

2.3.4 Hyperparameter Optimization

I explained in the section on validation (2.3.2) that during training, it can happen that the model is overfitting with respect to the training set. This is not a problem, as the model is not evaluated on the training set, but normally on the *validation set*. The results on the validation set can be used for example to experiment with various model types, and with different parameter numbers, as well as different hyperparameters. The latter refers to variations of the method to build the model, like preprocessings, the snippet length or the learning rate of a neural network. In practice, *combinations* of preprocessings and algorithms may be important as some may perform well together, while others do not.

The iterative process of modifying hyperparameters, then training a model, validating it and using the information to start again is called *hyperparameter optimization*. As by iterating again and again, using the same validation set, it is possible that the hyperparameters are overfit with respect to the validation data. This can only be verified with another independent set, often referred to as *test set*. When not considering hyperparameter optimization the terms 'test set' and 'validation set' are often used synonymously.

Avendi [6] describes "the most shameful fault of a data scientist" to allow information of the test set leak into training, which would mean an overfitting to the test set. This defeats the purpose the definition of the test set. He gives an example from a competition on Kaggle [73], where some competitors show extremely different performances on their given data, and on independent data. One model achieves rank 10 on the given data, but only rank 3165 on new ones.

2.3.5 Naïve Bayes

I wrote the following subsection in 2013 already, for my Master's thesis [33]. I only made small adaptations for the version shown here.

Bayes classifiers use the probability theory based on the work of *Thomas Bayes* for their classification. Well explained introductions to Bayes classi-

fiers can be found in [14, ch. 6.1] and in [11, ch. 8.2]. Given a domain of classes, $\mathrm{dom}(C) = \{c_1, \ldots, c_m\}$ and a set of attributes $\{A_1 \ldots, A_n\}$, an object instantiation can be written using the attribute values $(a_1, \ldots, a_n)$ of the attributes $A_1, \ldots, A_n$. The naïve Bayes classifier computes the conditional probability $P(C = c_i \mid A_1 = a_1 \ldots, A_n = a_n)$ for all classes c_i. The class with highest probability will be predicted. In practice it is not possible to store all probabilities for all possible given values, because their number increases exponentially with the number of attributes n. Applying Bayes' rule changes the calculation to

$$P(C = C_i \mid A_1 = a_1, \ldots, A_n = a_n) = \frac{f(A_1 = a_1, \ldots, A_n = a_n \mid C = c_i) \cdot P(C = c_i)}{f(A_1 = a_1, \ldots, A_n = a_n)}.$$

The denominator is only dependent on the object; when using this formula for one object and all classes, in order to find the most probable class, the object is constant and so the denominator can be ignored. Otherwise, the denominator is needed in order to create proper probabilities. Using the chain rule and assuming conditional independence results in the formula:

$$P(C = C_i \mid A_1 = a_1, \ldots, A_n = a_n) = \frac{P(C = c_i)}{p_0} \cdot \prod_{j=1}^{n} f\left(A_j = a_j \mid C = c_i\right).$$

At this point $p_0 = f(A_1 = a_1, \ldots, A_n = a_n)$ is a normalization constant, $P(C = c_i)$ describes the empirical probability which is stored for every class. The conditional probabilities $P\left(A_j = a_j \mid C = C_i\right)$ for a symbolic class attribute A_j can be stored in a table. For numeric real value attributes, there are two possibilities: numeric attributes can be discretized and then handled like symbolic ones, or the probability density can be stored. In applications assumptions can be made for some continuous parametric distributions: A normal distribution is often assumed such that only the expected values $\mu_j(c_i)$ and the variances $\sigma_j^2(c_i)$ need to be stored.

Note that although a Bayes classifier is meant to approximate the class probabilities, it is usually not evaluated how well it actually produces these probabilities, but by accuracy. When for example the predicted, most probable class only has a probability of 30%, it is still predicted, because of the lack of alternative probable possibilities. In practice, this behavior often leads to surprisingly good results for classification, although the probability estimations may be poor (cf. [160]). Naïve Bayes classifiers can be interpreted as simple Bayesian networks, which allow to weaken the conditional independence assumptions [14].

2.3.6 Linear Discriminant Analysis

The idea of the linear Discriminant Analysis (LDA) was described already in 1938 by Fisher [44], which is why it is also called *Fisher's Linear Discriminant*. It is a powerful method, but uses strong assumptions on the data. I describe LDA for two classes first and extend the explanation to more classes later.

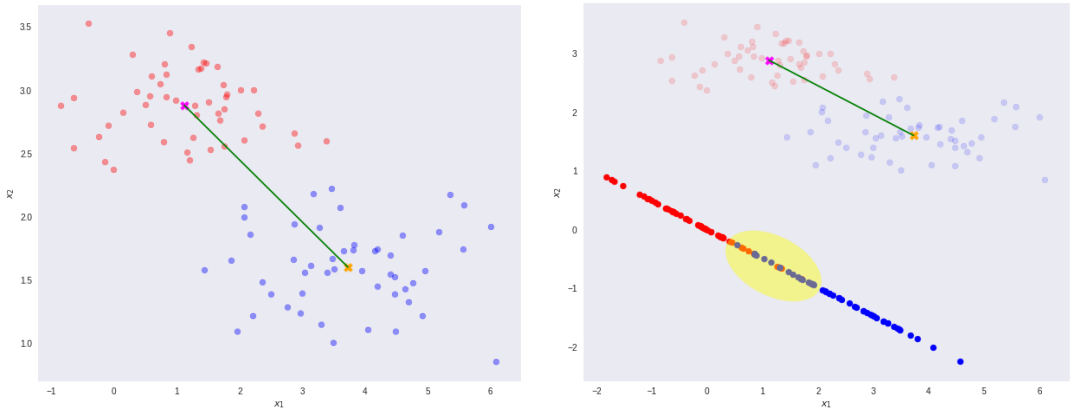


FIGURE 2.13: Visual example of Linear Discriminant Analysis, left with mean values connected by vector W, right with additional projection on W (Source: [133])

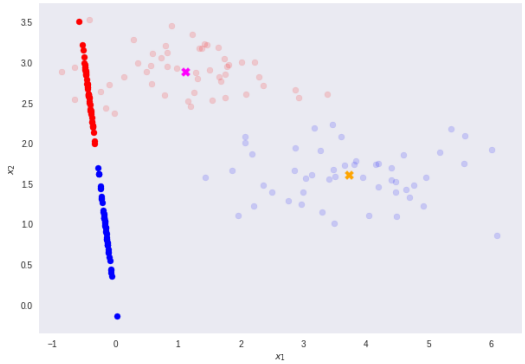


FIGURE 2.14: Example of LDA, projected on optimal vector W

The LDA assumes that the data of both classes are normally distributed with different means and identical covariance matrices. Silva [133] gives a great visual explanation, which I use in the following.

Figure 2.13 shows (left) points belonging to the red and to the blue class and their mean values indicated by crosses. Their mean values are connected by a vector W. When the data points are projected to W, as shown on the right side, this creates areas where the classes are perfectly separated and an area indicated by the yellow ellipse in which they overlap. Fisher's idea is to create a projection, which is not limited to using the mean values. He creates a large separation between the projected mean values while also minimizing the variances within each class and thereby minimizes the overlapping area, as shown in Figure 2.14. The math behind this projection is also explained in the work of Silva [133], more details are given in the work of Bishop [13].

I am not interested in the dimensionality reduction, but only in the result of the classification. In such a case, there are two common approaches extending any two-class classifier to multiple classes: First, one can iteratively train one class versus all other samples, that are combined into a pseudo-class to create a total of c classifiers, where c is the number of different classes. In order to gain a result for a new data point, the results of all classifiers need

to be combined. Second, one can train a classifier for each pair of classes. Again, the results of the classifiers need to be combined to get a result for a new data point.

The linear discriminant analysis was developed for several independent dimensions. Multiple consecutive time points are therefore considered to be independent, although this assumption is not given, since the order of the time points is ignored. In order to apply EEG data as multivariate time series, the data needs to be reshaped. This means that two pieces of information are lost during this modelling: First, the time dimension is treated as if the dimensions were completely independent. This is in contrast to the knowledge that the signal often changes only slightly over time. Second, the information that the data are taken from different channels is also lost. Consequently, I assume that these models do not perform well on this kind of data. However, I try these models to build a broader basis for a comparison with other models.

2.4 Neural Networks

Neural Networks are predictive models, which use learning samples to iteratively adapt their parameters. A layered neural network usually consists of an input layer, one or more hidden layers and an output layer. Each layer can be interpreted as a transformation into a new feature space. The number of dimensions and the kind of mapping are determined by the network's designer, while the concrete transformation is automatically learned during the backpropagation process that minimizes the prediction error (see e.g. [84]). I first give a brief historical view on the development of neural networks and in the later sections highlight specific aspects of them.

Artificial Neural Networks were invented already in the 1940s by McCulloch and Pitts [100]. They are based on the Perceptron, known as McCulloch-Pitts-Neuron. It is inspired by the biological neuron, which is the most frequent cell inside a humans brain. A neuron receives signals, which may originally stem from organs or other neurons. When sufficiently many signals excite the neuron it 'fires', meaning that it sends a signals itself. There are also inhibitory signals, which reduce the neurons excitation.

This concept was mathematically formalized such that a neuron has n inputs, and for each input x a weight w. Negative weights model inhibitory signals, while positive ones are excitatory. The weights' absolute value measures how much its corresponding input influences the neuron at all. Further, each neuron has a threshold value θ. When the weighted sum of the inputs exceeds the threshold value, the neuron sends the signal, otherwise not.

$$y = \begin{cases} 1, & \text{if } \sum_{i \in n} w_i x_i \geq \theta, \\ 0, & \text{otherwise .} \end{cases}$$

This formula fits the intuition of a simplified biological neuron. Mathematically, it describes a half-space in the n dimensional input space. If the input lies on one side of the plane $\sum_{i \in n} w_i x_i \geq \theta$ (or exactly on the plane) the neuron fires, otherwise not. Considering the task of classifying inputs, from two classes, then problem instances can only be classified perfectly (without any errors) if the classes are linearly separable. Other problem instances cannot be solved by one neuron. Also, more than two classes cannot be handled by one neuron, since the output can only by 0 or 1.

However, several neurons can be combined to build a *neural network*. Hornik showed [65] that multilayer feedforward networks (layers are explained in Section 2.4.5) are not limited by linear separability, when bounded activation functions are used (see Section 2.4.3). Even with only one hidden layer any given function can be approximated with arbitrary precision, which is known as the *universal approximation theorem*. Unfortunately, the number of neurons which may be necessary to achieve this precision might be exponentially large.

Lu et al. [94] considered layered ReLU networks with $n + 4$ neurons per layer and were able to prove that the universal approximation theorem holds if the number of layers is allowed to grow.

Academic examples are often manually analyzed, to construct a network structure and the corresponding weights to solve the given problem. In practice, however, problems are too big to analyze them. There, an arbitrary number of (possibly high dimensional) samples is given for which it is unknown if there exists a perfect solution. So the problem instance is given to the algorithm to train, looking for a solution. In practice, it is not even clear what defines an *optimal* solution: It is simple to construct an algorithm that memorizes all training samples and their corresponding outputs such that the training set is perfectly fit. But one is actually interested in an algorithm that is able to *generalize* correctly from the training samples, which for adversarial data may even be impossible.

So, how to achieve such an approximation if the problem is too complex to analyze it? First weights and threshold are randomly initialized. Then, for each input, the error and the gradient of the error function is computed. This can be used to compute which weights and thresholds along each path from input to output contribute to the error and further manipulate these parameters to decrease the error. This procedure is called *backpropagation*. When it is iteratively applied, over time the error is reduced.

2.4.1 Training Process

The training process works as follows:

1. Take a (non-empty) subset t from the training data set. The data points currently considered are called *batch* of training data. Using the entire training data set as t is the default, called *Vanilla* gradient descent.

2. For each sample in t:
   a) Compute the models predictions
   b) Compare the prediction with the correct value and compute the error the model makes on the batch.
   c) Further, compute the gradient of the error function at the point given by weights and threshold values. This can be done by summarizing over the partial gradients obtained from the batches for each weight and each threshold. Note that partial gradients of t samples may yield different partial gradients.
3. Compute the sum of these partial gradients
4. Multiply the gradients with the learning rate
5. Apply the change to weights and thresholds
6. repeat

The standard version is *batch* training, which means that t is the full set of *all* training samples. When a predefined maximal of iterations is reached, or when an iteration has changed the model only marginally, the training can be stopped. The model has reached a point of a minimal error. This minimum is considered to be the solution of the training problem. Unfortunately, it can not be guaranteed that the found minimum is the smallest possible minimum. In other words, it is unknown whether it is a local or a global minimum.

When t contains only one element, the procedure is called *online* training. Normally, t is sampled without replacement in order to guarantee that all samples are used similarly often. Once there are no more samples remaining, a so called *epoch* ends. In the next epoch samples can be taken again from the whole training set. Typically at the end of each epoch the model is validated by applying a subset of the validation set to the model. Again, the training ends, when a given maximal number of episodes is reached or the validation loss (see Section 2.4.4) is smaller than a given target value.

Computing the true gradient with *vanilla* gradient descent is costly due to the large number of training samples. As alternative of using all training samples is to randomly sampled subset t, which is called mini-batch. The size of t needs then to be given by the *batch size* parameter. The corresponding training process is called stochastic gradient descent. This has the advantage that fewer computations are made between the weights and thresholds are adapted. However, this comes at the cost that not the true gradient, but only an approximation of the gradient generated from the current mini-batch is used and consequently not the full loss function, but only a partial sum is used. Obviously, these partial gradients may differ for different subsets and for the whole data set. During stochastic gradient descent one does not evaluate the true, but only a partial cost function.

Although this seems to be a disadvantage, this is also hypothesized to be one reason for the success of stochastic gradient descent in practice. The full error function is known to have local minima, in which a vanilla approach might get stuck. By using varying partial error functions, which likely have local minima at different positions, the chances are higher to finally reach a smaller loss.

There are applications with millions of data samples or even cases, where samples can be generated on demand. In such cases it may take too long, or it may even make no sense to use the above definition of an epoch. Instead, a fixed number is given that defines after how many batches the epoch ends. This is only relevant, when a network is evaluated on the validation data during the training, for example when *early stopping* is applied (see Section 2.4.13).

2.4.2 Optimizer

Gradient descent is among the most popular algorithms to perform optimization, especially to optimize neural networks. Ruder [124] gives "An overview of gradient descent optimization algorithms". During gradient descent, a loss function (or objective function) is minimized by updating the parameters stepwise in the opposite direction of the gradient of the loss function. The learning rate defines the size of each step taken. The typically used analogy is a ball rolling down a hill, following the slope until it comes to stop at a (local) minimum. Nice dynamical visualizations for several optimizers can be found in the post by Hansen [56].

One adaptation does not rely only on the current gradient, but also considers, similar to the ball, the former gradient to adapt a momentum term. This accelerates the training speed when the gradient in one direction is similar in consecutive steps and dampens oscillations when the gradient has changed. One improved version of momentum is the Nesterov accelerated gradient. It uses the momentum to approximate the position at the next step and computes the gradients for the approximated future position.

Two newer approaches Adadelta [159] and Adam [80] (Adaptive Moment Estimation) compute a gradient normalization scheme for each parameter based on the former gradients. Adadelta stores a variable that holds an exponentially decaying value of the past squared gradients, which is used to adapt the parameter-specific normalization. Adam stores in addition to the exponentially decaying average of the former *squared* gradients another variable for the exponentially decaying average of the past gradients, which is used to compute the normalization factor. Note that these normalization methods are also sometimes described as computing parameter-specific learning rates.

When I started to run the experiments for this work, I also tried different optimizers. I found that Adadelta and Adam performed better than the classical

stochastic gradient descent and thus use them for my experiments. Since Adadelta is the simpler method, I used it most of the time and only sometimes tried Adam, which in my experiments always performed similarly.

2.4.3 Activation Functions

In the original work of McCulloch and Pitts [100], the Heaviside function was used to model a neuron (from the beginning of this chapter) "firing". With $x = \sum_{i \in n} w_i x_i - \theta$:

$$H(x) = \begin{cases} 1, \text{if } x \geq 0 \\ 0, \text{otherwise} \end{cases}$$

However, to be able to train neurons by gradient descent all applied functions between input and output need to be differentiable and the gradient is to be used [84]. For the Heaviside function, the gradient is either undefined or zero. Therefore, it is not well suited for the backpropagation process. An alternative function, that can be interpreted as a "smoothed" Heaviside function is the so called *logistic function*, which has the nice properties that its co-domain is the interval $]0, 1[$ and it is easily differentiable:

$$\sigma(x) = \frac{1}{1 + exp(-x)}$$

If negative values are to be allowed for a result, the hyperbolic tangent (tanh) function may be used as activation, since it allows values between -1 and 1:

$$\tanh(x) = \frac{2}{1 + exp(-2x)} - 1 = 2 \cdot \sigma(x) - 1$$

In 2009, Jarrett [75] investigated several rectified functions, including one function which is since then more and more used as activation function. The *Rectified Linear Unit* (ReLU) uses:

$$\text{ReLU}(x) = max(0, x)$$

The properties of this function are mathematically less 'elegant'. It is unbounded, and thus does not satisfy the conditions of Hornik's proof [65] explained at the beginning of this chapter. It returns zero for any input smaller than zero and thus does not distinguish between different negative inputs. However, the derivative is extremely simple to compute and despite their simplicity the results with ReLUs are often very good (see e.g., [103]). Since their derivative is 0 or 1 one can expect it to be on average $\frac{1}{2}$, which is more than the maximal possible value $\frac{1}{4}$ reachable by tanh. All these functions are illustrated in Figure 2.15.

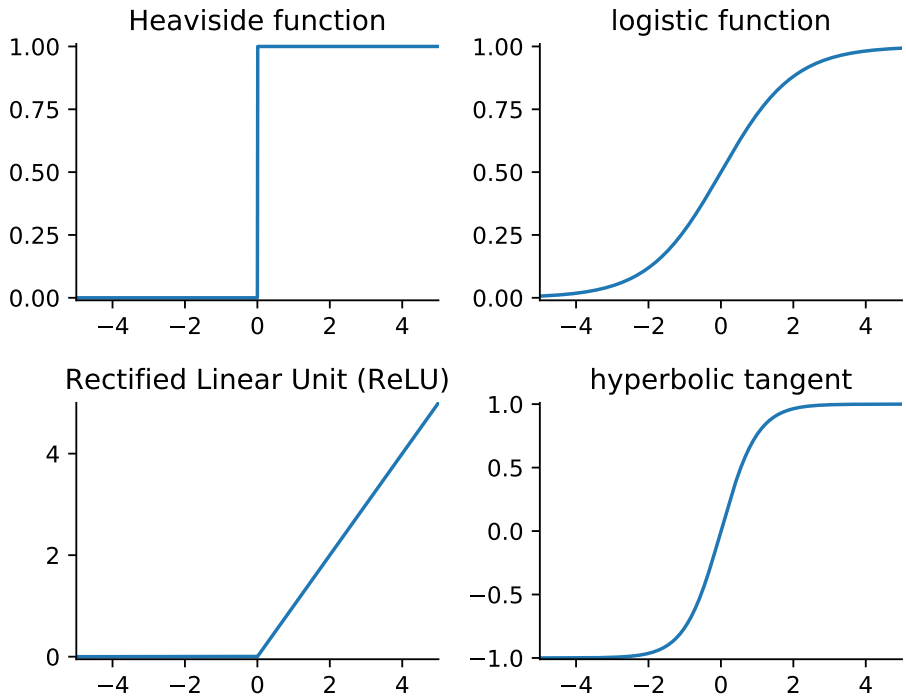


FIGURE 2.15: Frequently used activation functions

Softmax

Gómez explains the *Softmax* function nicely [19]. The previous activation functions consider the inputs of one neuron and compute this neuron's output. In a classification task, where all samples belong to exactly one of c classes, it is desired that one can interpret the network output as probabilities for each class. Therefore the outputs need to be normalized to sum 1. Since this normalization does not change which neuron yields the highest activation, one can afterwards still predict the class with the maximal value using the *argmax* function. Unfortunately, the argmax function is neither continuous nor differentiable [49, p.180ff], which is necessary to perform the error back propagation in the learning process. The *Softmax* function follows the idea to create a continuous and differentiable function which serves this purpose, like the logistic function is a continuous and differentiable approximation of the Heaviside function. Obviously, it needs the inputs of all neurons:

$$\text{softmax}(s)_i = \frac{\exp(s_i)}{\sum_{j=1}^{c} \exp(s_j)}$$

The output vector has the desired properties of probabilities, they are all between 0 and 1, and their values sum up to 1.

2.4.4 Loss Functions

The loss function defines how the errors of several samples are summed up into one value, determining the model quality. As explained before, it is mandatory to use differentiable functions, preferred with gradients mostly differing from 0. For regression tasks, the Mean Squared Error (MSE) can be used as loss. For a single data point, it can be computed as the name tells, by computing the mean squared difference of the true value t_i and the predicted value s_i for each of the d dependent variables.

$$MSE = \frac{1}{d} \cdot \sum_{i=1}^{d} (t_i - s_i)^2$$

For a classification with c classes, *one-hot encoding* is normally used. This means that the vector $\vec{t}$ contains c values: 1 for the true class and 0 otherwise. The cross-entropy loss is mostly used for classification:

$$CE = -\sum_{i=1}^{c} t_i \cdot log(s_i) = -log(s_p) \quad \text{with} \quad p = \underset{i=1,\dots,c}{\arg\max}\, t_i$$

This works especially well when combined with a Softmax function in the last layer [19]. Almost all terms of the sum are zero such that the only remaining summand is the one for the true class t_p, it can thus be written:

$$CE = -log\left(\frac{\exp(s_p)}{\sum_{j=1}^{c} \exp(s_j)}\right)$$

One loss function that is not commonly used is the cosine similarity. It computes the angle between two vectors. However, Barz and Denzler [7] recommend to use it, especially for small data sets.

$$cossim = \frac{\sum_{i=1}^{c} t_i \cdot s_i}{\sqrt{\sum_{i=1}^{c} t_i^2} \cdot \sqrt{\sum_{i=1}^{c} s_i^2}}$$

This can be simplified because of the one-hot encoding. Again, almost all terms of the sum in the numerator are zero such that the only remaining summand is the one for the true class t_p. Similarly, for the first sum in the denominator only one summand $t_p^2 = 1 \cdot 1$ remains, so that the sum and its square root can be omitted. Note that the Softmax function normalizes the values to sum up to 1, while the normalization here uses the *squared* values.

$$cossim = \frac{s_p}{\sqrt{\sum_{i=1}^{c} s_i^2}}$$

Motivation of using the Cosine Loss

For neural networks in classification tasks, Barz et al. [7] suggest an alternative to the typically used softmax function in the last layer which is normally combined with a categorical cross-entropy loss. They perform experiments on 5 'small' data sets containing pictures, where they define a data set to be small if each class has fewer than 100 training samples. Additionally, they use two bigger data sets, CIFAR-100 and AG News, for which they use subsampled versions with different numbers of samples per class. The used data sets use 67 to 555 classes and have at least 2000 training samples. One of the tested data sets contains texts, all others use images.

The used processing is highly specialized to work on pictures, with methods shown to be successful, there: The network structure is the common ResNet-50 architecture for the small picture data sets. As data augmentation, they randomly flip images horizontally and they randomly erase parts of the pictures. Stochastic gradient descent is applied as optimizer with a specialized schedule to change the learning rate in a pre-defined manner over time. This schedule uses several cycles with increasing lengths, where for each epoch the learning rate is reduced from a maximal to a minimal value. While the minimal value is always fixed to 10^{-6}, they perform experiments with 8 different maximal values ranging from 0.001 to 2.5. Finally, they report the best performance achieved after any epoch.

For this specialized processing, and 10 runs per experiment, Barz et al. [7] illustrate for their small data sets that the cosine loss performs significantly better than classical softmax with cross-entropy. The effect is not significant for the big data sets.

As they only give a comparison on different processing methods, it is okay not to use an independent test set, but to test on the validation set. However, this processing does not allow a generalization to other data sets, as it is possible that the tried methods vary in their ability to overfit to the test set during the training. Further, it is possible that their effect is influenced by the network structure, it may only work for stochastic gradient descent or only when also combined with a learning rate schedule. As only 10 runs are performed, and only the average of these runs is presented, it is also possible that there are strong fluctuations between the single runs. Thus, it is questionable whether or not the cosine loss should be *generally* used instead of the softmax with cross-entropy.

Their experiments use data sets with far more classes than the ones I consider in my work (in Chapter 5 and Chapter 6) and the number of independent samples is much smaller, in my case. I assume that their training space is better covered due to the higher number of samples and their applied data augmentation. However, it is worth to try the cosine loss, instead of categorical cross-entropy.

> **Q2:** *Which loss function performs better? The classical combination of softmax and categorical cross-entropy or the cosine loss?*

2.4.5 Layers

Dense

I did not yet specify how neurons are connected inside the network. Since neurons only process information in one direction they form a directed graph. This graph is called *acyclic* if it does not contain any cycles or loops. One of the most common structures for neural networks is a layered structure. In a layered network all paths from any input neuron to any output neuron have the same length. An equivalent definition states that there are only edges between subsequent layers. This means that each path from any input neuron to an arbitrary neuron has the same length (or distance). The set of neurons with the same distance to an input neuron is called a layer. If all neurons with distance k are connected to all neurons with distance $k+1$, then these two layers are densely connected. In this case every output of the neurons in the current layer is an input for each neuron in the following layer.

One generalization of the formular of a neuron "firing" is to use an arbitrary activation function *act*. The output of a neuron i in layer j is then:

$$x_{i,j} = \text{act}_l \left(\sum_{i=1}^{n} (w_{l,j,i} \cdot x_{l-1,i}) - \theta_{l,j} \right)$$

Because each neuron in a dense layer has weighted connections to each neuron in the preceding layer, dense layers have many parameters and therefore a high computational power. But they are also prone to overfitting the data.

Convolution

Suppose a neural network was used to find certain features, like a dog, in pictures. Using a dense layer, it would not only learn how the dog looks, but also its exact positioning from the training data, as both pieces of information are encoded within the weights. For data similar to the training data, i.e. shifted toward an arbitrary direction, the network would be unable to detect the dog. This motivates a different type of layer, that is able to find features that are invariant toward position, the *convolutional* layer. It applies the convolution operation, which takes two signals to compute an output signal [137]. A convolutional layer uses the network input image and a fixed set of weights (here also called *filters*) to create location-invariant features. Therefore, the input image is regularly cut into sub-pictures, on which the same filters are applied.

In the following, I give an example of how convolution is applied by using a fixed filter, the so called *Sobel filter*, that uses the two matrices G_x and G_y (see e.g., [126]), which are applied on an input image A.

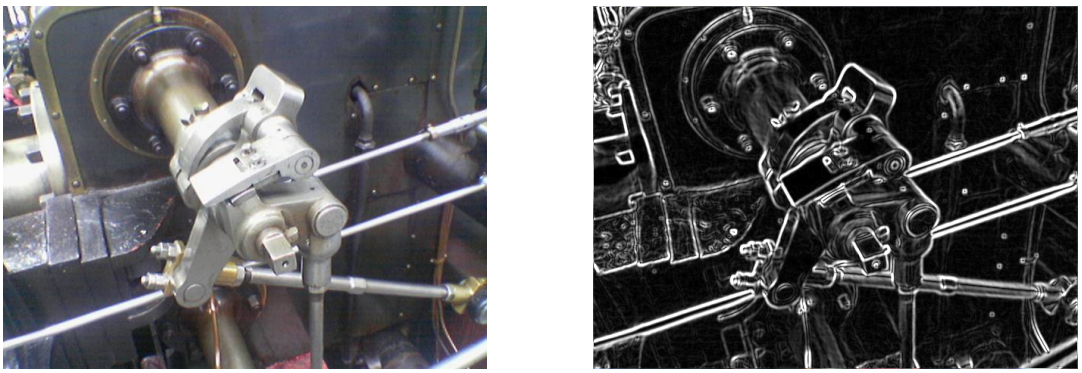


FIGURE 2.16: (left) picture of a valve, (right) the same picture after the Sobel operator has been applied (Sources: [68], [69])

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & +2 & -1 \end{bmatrix} * A$$

A picture can be encoded as an array of values where each value corresponds to the color of a specific pixel. When G_x is applied to an image, it results in high values when the right part is brighter (higher values) than the left part. Very high or low brightness of the resulting image represent locations with vertical edges in the original. Similarly, G_y detects horizontal edges. To create a matrix that shows edges in arbitrary directions G_x and G_y are typically combined to $G = \sqrt{G_x^2 + G_y^2}$, where the squares mean element-wise multiplication. The resulting picture shows edges in any direction as shown in Figure 2.16. Generally, the output of a convolution shows the pattern's strength of activation for each local part of the input.

Both dimensions, height and width, of the picture are similarly important, which is why two matrices G_x and G_y are necessary, which are both moved in both directions vertically and horizontally over the image. With *two* directions (dimensions) this is a 2D convolution. Note that the 2D does neither refer to the input space nor to the dimension of the filter (which are both 2, in the example), but to the number of dimensions in which the filter is moved.

Dumoulin and Visin published in 2016 "A guide to convolution arithmetic for deep learning" [41]. This work does not only describe convolution and pooling (see Section 2.4.6, but also thoroughly explains details with many figures, which are publicly available on github[3]. I recommend it to the interested reader and I use some of the pictures in the following.

Figure 2.17 depicts an input feature space of size 5×5 in light blue — which corresponds e.g., to a grayscale image — across which a filter of size 3×3 slides. The filter values are the small numbers in the dark blue areas. At each location, where image and filter overlap, their product is computed and the

[3]https://github.com/vdumoulin/conv_arithmetic

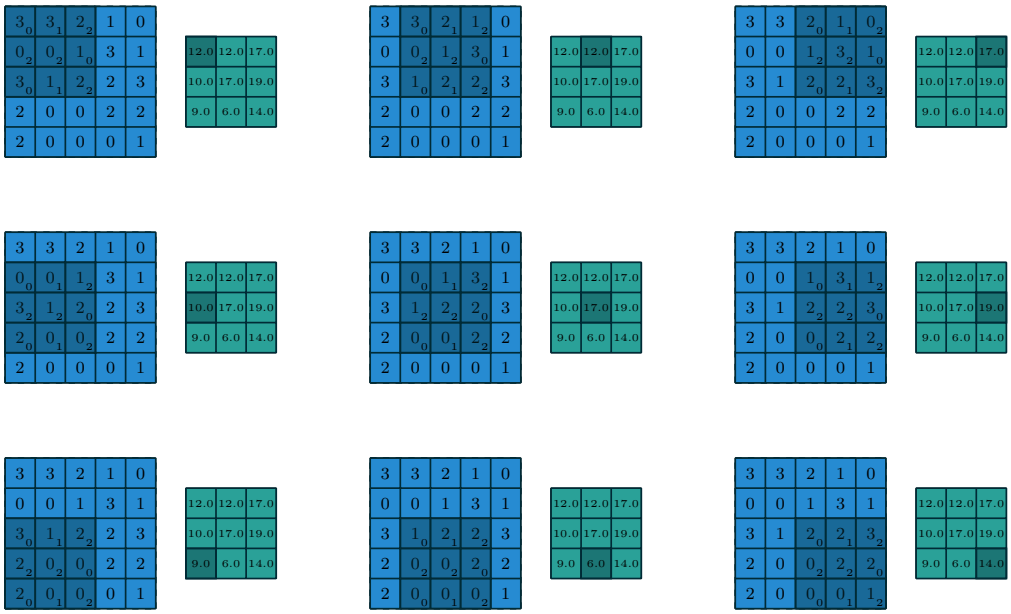


FIGURE 2.17: Applied 2D convolution on 2D feature map (Source [41])

results are summed up and shown in the feature map on the right, which has size 3×3. The highlighted darker regions correspond to the filter location on the left such that the computations can easily be verified.

The filter cannot be applied at the border of the image if parts of the filter would not lie inside the image. If the borders of the image are also important, then one can add additional zero values at the beginning and at the end of an axis. The parameter that specifies the number of added zeros is called *zero padding*.

The first successful applications of convolutions in neural networks stem from 1990 for images as input. LeCun et al. [88] used them to automatically digitize handwritten zip codes. Lawrence used them for face recognition [86] in 1997.

One possibility to decrease the size of the resulting output feature map and to save computation time is to use a higher *stride*. This is the distance between two consecutive positions for which the filter is applied. It can be defined independently for both dimensions of the 2D convolution. If the example from Figure 2.17 was modified to use strides of 2 for the x-axis and the y-axis, then the middle row and middle column of the result would not be computed and the size of the output feature map would be 2×2.

When the given picture is not grayscale but for example uses RBG (red, blue, green) as color encoding, this adds a third dimension. Then there are: height, width and color channel. The applied filter is then by convention also three-dimensional, such that the filter again is moved only in the dimensions height and width. As the convolutions computes the sum of the involved

values, applying a three-dimensional filter creates a two-dimensional output. To generate a three-dimensional output feature space, one uses several filters, whose application builds the channels of the resulting output space.

The data considered in this thesis stem from neural sources and have similarities with pictures. The input consists of data from several channels, measured at different points in time. Thus channels and time build the two dimensions. In contrast to pictures, these dimensions model different aspects of the data. Practically, the channel's location on the scalp would be helpful, but there is no simple way to include this information in the neural network. It is not possible to preserve the location on the scalp in a one-dimensional ordering of the channels. In contrast to the processing of pictures, which can be natively done by using 2D convolutions, Neural data — interpreted as multivariate time series — are naturally processed with 1D convolutions, where normally a 2D filter is applied, that is moved along only one dimension: time.

Note that the convolutions, as explained here, apply filters that look for patterns that are *local*, which means that they have the implicit assumption that there are helpful local patterns within the data. One version, which allows to look for patterns more widely spread in the signal is to use *dilated* convolutions [158]. There, the filters are applied on the image with a stride such that the found patterns relate to a more widespread area.

2.4.6 Pooling

Besides from using strides within convolutional layers, *pooling* operations can be applied to reduce the size of a feature space. This means a pooling function, like minimum, maximum or average is applied in order to aggregate information. Note that the most common pooling function is the maximum. This is plausible, when interpreting convolutions as feature detectors. Then the highest value represents the strongest occurrence of the pattern of the filter in the input space. Then the Max-Pooling operation loses local information on where the pattern was found, but focuses on the strongest occurence. The combination of convolution and Max-pooling was very successfully applied [83] e.g., if one is only interested in which object is in a picture, but not where this object is. However, if one would be interested in patterns not somewhere, but everywhere in a picture, then pooling with the Minimum function (short Min-Pooling) might be a better choice. I hypothesize that such a filter might be helpful when detecting special kinds of noise, artifacts or manipulations in pictures.

Figure 2.18 depicts a 3×3 Max-Pooling applied on a 5×5 input space with stride 1 for both axes. The input space is again shown on the left and the output on the right. Note that the default *stride* is not 1, as seen in the Figure, but the pooling size. Then it is intended to keep only the maximal values and discard the others. When pooling is applied with padded zeros, then the size of the resulting image would be bigger.

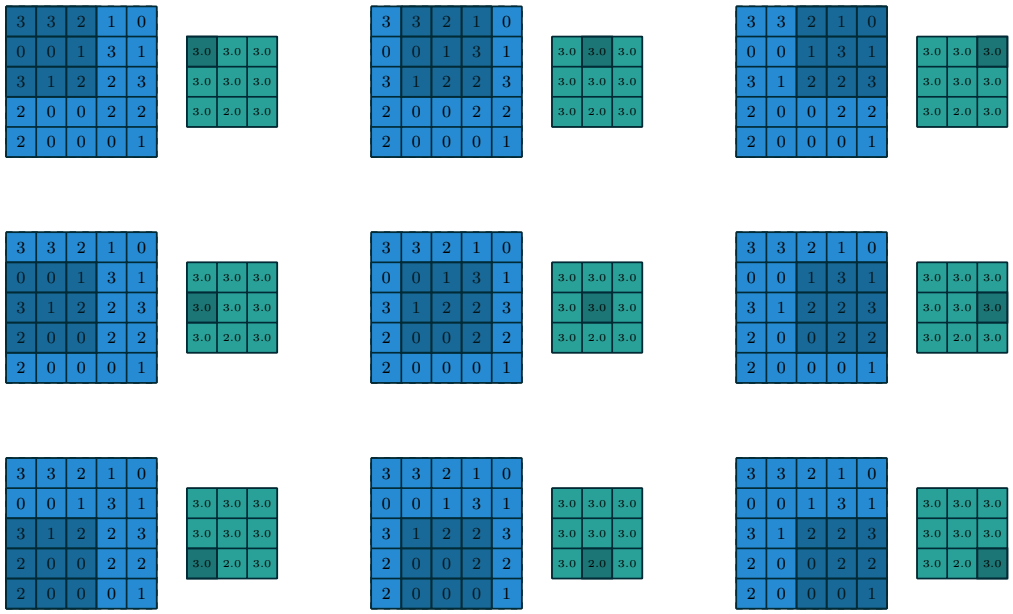


FIGURE 2.18: Applied Max-Pooling on 2D feature map (Source [41])

In applications within convolutional neural networks, so called *convolutional blocks* are often used. They consist of (at least) one convolution layer, followed by a Max-Pooling layer. An alternative form of a convolutional block uses Max-pooling as alternative to strided convolution [138]. In image processing the pooling size and the stride are often 2, such that the pooling layer creates an output that looks similar to the original image, just compressed by a factor of 2 in both dimensions. It is also typical to use increasing numbers of filters, which results in a constant number of weights per block. As these blocks and also the pooling operation are iterated, it is helpful to have the input dimensions to be multiples of 2, or best even the powers of 2. Otherwise, one has to handle the problem that input and output dimension do not perfectly fit. At the borders of the feature space, one has to decide whether to use padding to add zeros, or to pool 3 instead of 2 values. Note that this problem shows up again in Section 6. There, the input length is 250 and Max-Pooling is applied iteratively in one dimension with a factor of 2, which induces the described pooling of factor 3 at the borders of the feature space.

2.4.7 Vanishing Gradient Problem

During training in some layered networks a problem shows up when regarding the weights of the early layers a layered network, during the training process: in some cases the weights hardly change at all; in other cases they change with huge speed. This is caused by the backpropagation process, where the gradients of the later layers are used to adapt the earlier ones. When one layer is processed it influences the (average) gradient of the previous layer by a factor. If this factor is smaller than 1 for several layers, then all

of these factors reduce the gradients and thus the training speed in all earlier layers. Thus, this effect is called *vanishing* gradient problem. The opposing effect — weights that change with a huge speed — occurs if several factors are greater than 1 and is called *exploding* gradient problem. Networks with more layers are obviously more prone to these effects, since they have more factors.

Hochreiter analyzed the these problems thoroughly [63], I consider this problem one of the most severe issue with neural networks. Since its discovery in the early 1990s, many researchers have worked on this problem and approached it differently. In the following I will describe some of these approaches.

2.4.8 Weight Initialization

One approach of coping with vanishing or exploding gradients is to consider their variance. Experts agree on the fact that, for optimal training speed, the weights should be initialized around zero. The question on the variance seems to be more difficult, and therefore several variants have been proposed. In general they can be based on a normal distribution or on a uniform distribution, like the distributions based on the work of He et al. [59] (*he_uniform, he_normal*) or the distributions based on the work of Glorot and Bengio [48].

With certain assumptions, the variance of weights in one layer of a layered network, can be computed from the variance of earlier layers. Peruncic [115] motivates the desire for similar variances across the activations of each layer in order to achieve a weight adaptation with similar speeds in all layers which avoids vanishing and exploding gradients. He finds that the weights variance should be chosen with respect to the number of neurons n_i in the layer i and the activation function. Glorot and Bengio [48] recommend to choose $Var(w^{(i)}) = \frac{2}{n_i+n_{i+1}}$ for the typical tanh activation. Since ReLU returns zero for negative inputs it roughly removes half the variance [59], thus the optimal value for ReLU is $Var(w^{(i)}) = \frac{4}{n_i+n_{i+1}}$. Peruncic [115] illustrates the correctness of the result by using the famous MNIST data set, which consists of 60,000 scanned digits. He simulates the training and varies the variance of the weights during the initialization of a network with 5 hidden layers and 100 neurons per layer. Finally he evaluates the distributions of the neuron activation in each layer after training for 12 epochs to verify the correctness of the theoretical results. Since all layers use $n_i = 100$ neurons, this yields the standard deviation $\sigma(w^{(i)}) = \sqrt{\frac{4}{200}} \approx 0.14$.

I depict his result in Figure 2.19. The results verify the theory: A carefully chosen initialization can cause weight activations after training to be in the same magnitude. Thus, I use the proposed variance in all of my experiments in Chapter 5.

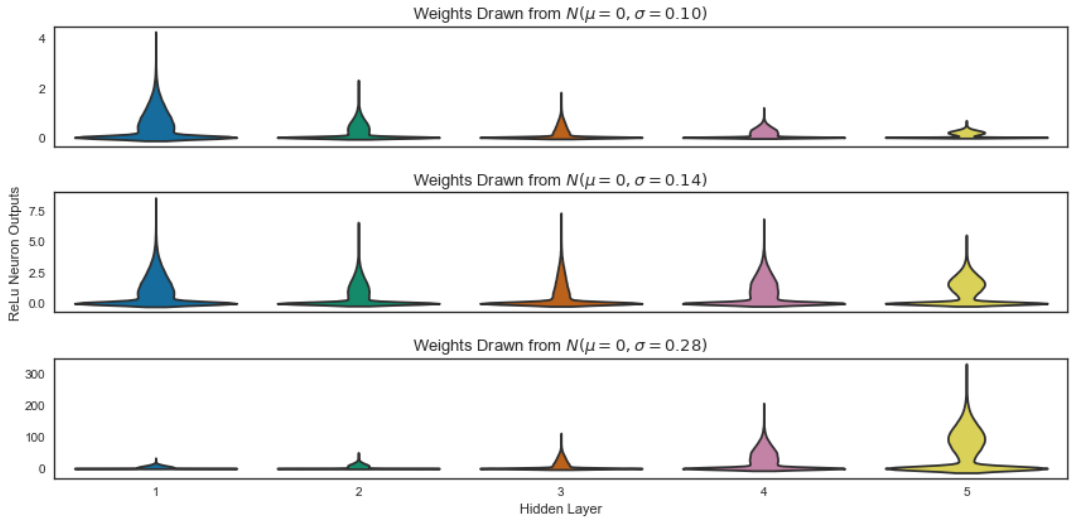


FIGURE 2.19: Output progression for ReLU activations (Source: [115])

2.4.9 Local Minima During Training

Swirszcz et al. [142] go even further; they analyzed local minima in training of neural networks and proved that a bad initialization can cause the network to converge to a local minimum. Additionally, they generated a minimal data set and performed experiments showing that a certain percentage of networks converge to a bad local minimum. They also compared these ratios for different activation functions and optimizers with the two findings: First, the logistic (sigmoid) activation function lead to significantly worse local minima than ReLU. Second, classical gradient descent performs better than the modern Adam optimizer (Section 2.4.3). Note that this effect of non-converging networks will be a practical issue with the data set in Chapter 6.

Although this effect is known for several years — the work of Swirszcz et al. [142] was published in 2016 — only few papers on the application of neural networks mention this effect at all. I suppose two possible causes: the effect occurs rarely in practice, or researchers only present the result from the best initialization, instead of the average to increase their chances of the paper being accepted for publication. There are good arguments for both, presenting the best or the average model. But in order to make results clear, comparable and reproducible ones should mention the number of runs from which the results were picked.

2.4.10 Batch Normalization

In Section 2.2.5, I explain that the input data should be normalized. A similar method, *batch normalization* (BN), can also be applied within the network, as the work of Ioffe and Szegedy [74] describes. It uses the mean and standard deviation computed for the previous batches to normalize the outputs of the current batch before the weight update. If this would be applied as just

described, it would limit what the layer can represent. Therefore, this is addressed by introducing two trainable parameters, which scale and shift the normalized value. These parameters are trained just like the other parameters during the training phase. However, when the network is tested, the moving average and variance that was estimated during training are fixed and used. This is necessary to ensure that the mapping between input and output stays unchanged during the evaluation of the model.

Implementation of Batch Normalization in Keras

At this point, I need to emphasize the implementation of the batch normalization in Keras [23], since it is the framework I use for most of my computations and it does not always work as expected. Vryniotis [150] describes this problem in more detail in his blog and even offers implementations that would solve the problem.

But what is the problem? In a pre-training approach, typically the first layers of a network are frozen to keep their pre-trained processing identical and only the later layers are trained with the new data (so called *transfer learning*. This is a very common approach for image processing, because many winning models of the Imagenet competition [29] are publicly available and object recognition often works better, when pre-trained networks are used, instead of randomly initialized ones. Before Keras' version 2.1.3, the implementation of batch normalization incorrectly kept updating the mini-batch statistics although it was *frozen*, where *frozen* was supposed to mean that the layers would be fixed and not changed during the training process. All of the parameters of of the batch normalization layer were adapted.

This issue was solved in Keras' version 2.1.4 by fixing the *trainable* weights when the layer is frozen. But batch normalization has two kinds of parameters: the two statistical parameters for mean and standard deviation that would still be adapted, and the two trainable parameters for scaling and shifting that would be frozen. Thus this problem is only resolved if data the data distributions of the first training phase and the second training phase are identical.

It is improbable that the data statistics of the new data set are identical to the original one. But nevertheless, during testing, the early frozen layers would switch back to the statistics trained on the original data set. Because of this difference between training and testing, the adapted later layers would receive differently scaled inputs and would thus not be able to perform properly. User reports show that models perform very differently, when the same data is given in the training and in the testing phase. There, I fully agree that this behavior cannot be intended and should be changed. However, Keras has recently been included in the tensorflow framework, where the problem is solved by not adapting any parameters, when a layer is frozen.

As one consequence for this work, I decided not to use pre-trained networks. Unfortunately, I had already made too many experiments to switch frameworks. With this lesson learned, I would *not* recommend to use Keras, but another framework like PyTorch [111].

2.4.11 Regularization

One approach to reduce overfitting is to use a *regularization* technique [105].

L1 and L2 Regularization

Very simple approaches add a penalty term to the loss function. Since greater weights cause greater gradients, and this might enable "exploding gradients", one can penalize the weights. The *L1 regularization* penalizes weights linearly by considering their absolute values. To increase the penalty for bigger weights even more, the weights' squared values can also be used, which is known as *L2 regularization*. L1 and L2 differ only in the relative strength of the penalty depending on the relative size of the weights. The latter is also called *weight decay*, since the penalty of the loss function results in a backpropagation process, in which in each iteration every weight is multiplied by a constant factor smaller than 1.

Dropout

Neural networks are known to be highly adaptive. When given a vector of n values, for which the output is correct, it may be possible that changing only one value may lead to a different output. In order to increase this robustness, the idea came up to ignore some of the input values during the training process. By randomly choosing which values are ignored, this so called *dropout* [139] is meant to increase the robustness and reduce overfitting. It was presented first on layered networks. It can be applied for example on the input layer to randomly ignore some proportion of the given values. The proportion can be controlled by the *dropout rate d*, for which 0 means no neuron is dropped, and 1 means that all of the neurons are ignored. Note that dropout can not only be used for dense layers, but on arbitrary layers.

For technical reasons, the implementation of dropout works by adding one layer to the network in front of the one where the dropout is to be applied. The dropout layer has the same number of neurons and connections exist only one per neuron with value 0 or 1, but this value is not fixed, but randomly determined. Whenever a batch is passed through the network during training, random numbers are generated for each of these connections. If they surpass d the connection is set to 1, otherwise to 0. During the testing phase, these connections are constantly set to 1.

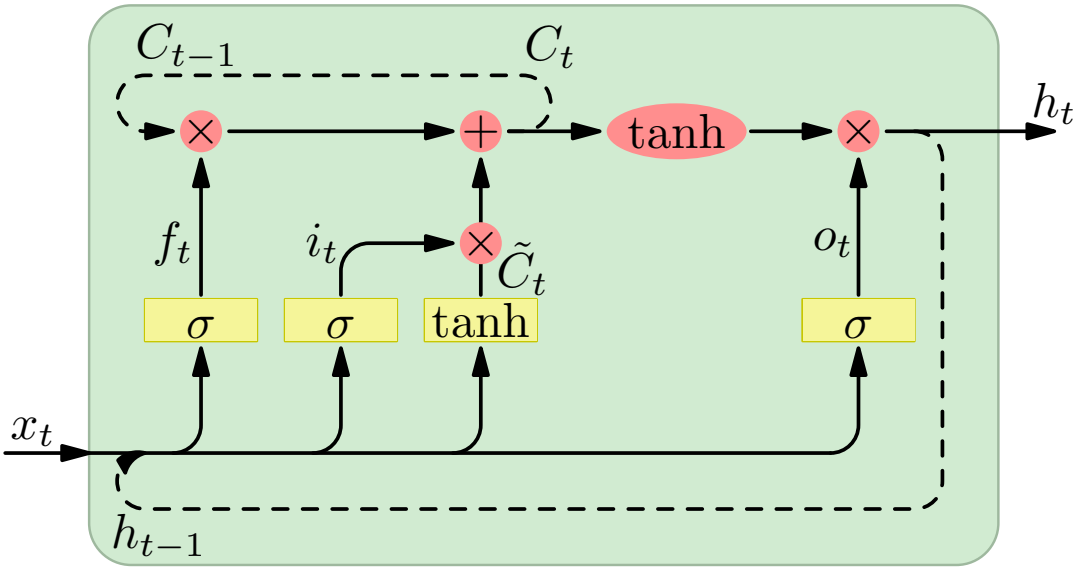


FIGURE 2.20: Long Short-Term Memory

Creating systematic discrepancies in weight activations between training and testing is a problem that needs to be handled carefully. For dropout, one can deal with this problem relatively easy: Since the average output of the dropout layer is reduced by a factor of $1 - d$ during training compared to the one during testing, the average activation is kept similar by multiplying each output by $1 - d$ during testing.

2.4.12 Long Short-Term Memory

The Long Short-Term Memory (LSTM) [64] was created by Hochreiter and his supervisor Schmidhuber in 1997. It is a recurrent network structure. Recurrent means that the graph structure has at least one cycle. Although LSTM is often referred to as *unit*, this is imprecise, as in fact it is more of a substructure rather than a single neuron. It is designed to process time series and has the ability to learn weights. In contrast to classical feedforward layers, the LSTM has the ability to memorize information over longer time. An overview is given in Figure 2.20.

Before I go into details on the processing of the LSTM, I first explain *gating*, which is an improvement first presented by Gers and Schmidhuber in 1999 [47]. Gates are illustrated by a red circle with the $\times$ symbol. This is a process to limit the values of a given vector by performing a pointwise multiplication with the gating vector, whose values are between 0 and 1. These values of the gating vector are gained by applying the logistic function σ, first. Note that all operations on vectors are indicated in red, while matrix operations are given in yellow. Normal forward connections are drawn through, while recurrent connections are drawn dashed.

The input consists of two parts that are always used together: the vector $\vec{x}_t$ with dimensions $m \times 1$ comes from the given data, and the vector $\vec{h}_{t-1}$ contains the output of the last time step $t-1$. The output dimension m is a parameter defined by the network designer, often referred to as 'units'. Using the output as new input is the first of the two recurrent connections of the LSTM.

This combined input is used to compute the *cell state* C_t and for three different gating processes: the forget gate f_t, the input gate i_t and the output gate o_t. Each of these processes uses a separately trainable weight matrix with dimensions $(m+n) \times n$ to learn which information to forget W_f, which inputs to use or ignore W_i and which values to output W_o. The cell state is kept over time by the second recurrent connection. Which part of C_t is forgotten, is determined by the forget gate. The second influence of the cell state is added to the current cell state in each time step and comes from the cell input activation vector $\tilde{C}_t$. This vector is generated by the corresponding weight matrix W_c and needs to pass the input gate first. After the updates of the cell state it is first limited to the range of [-1,1] by a hyperbolic tangent function. Then it is passed to the output gate. The resulting vector is the output for the current time step h_t. It is also transferred back to the input by the recurrent connection, where it is used in the next time step. Note that the cell state C_t cannot change exponentially over time, but only linearly due to the gating and the addition. The potential risk that the values grow to huge values over time is small, because of the forget gate.

In practical implementations, there are many options available for the LSTM: It is possible to not just return the output of the last state, but all outputs for each time step. Additionally, the cell state can be passed as output. Further options allow to change weight initializations of the matrices and the corresponding bias vectors, as well as the activation functions and also to add different kinds of regularization, for example dropout. If the given batches are ordered in the time dimension, the *stateful* option can be used in order to keep the last output of the previous batch or to set it to zero.

2.4.13 Callbacks

I use the term *callbacks* to describe various heuristic methods implemented by callback functions that try to cope with training problems by using information gained during training.

Early Stopping

One approach to limit over-fitting is to keep track of the model's prediction quality on the validation data set and memorize the best model regarding the validation data. A good overview of the optimal choice for the parameters is given by Prechelt [118]. He considered several criteria that can be used for

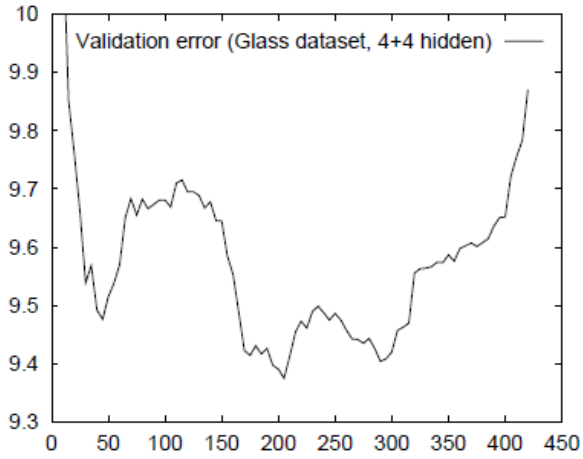


FIGURE 2.21: Error curve on a validation set (Source: [118])

early stopping and evaluated those criteria with respect to several parameters; among them are training time, efficiency, effectiveness and robustness. For example one could consider to stop the training as soon as possible, once the validation error increases. This would save the most training time.

Unfortunately, the training histograms differ between theory and practice. In Figure 2.12 (p. 29) I showed an example specifically chosen to explain theory. Prechelt's example from practice (Figure 2.21) shows random fluctuations over the training time and that the first increase of validation error is not necessarily the optimal stopping criterion.

His conclusions are that criteria that stop the training later lead on average to better results. One negative aspect of these criteria is that the training time needed is often large and can vary dramatically. He recommends: Fast stopping criteria should be used unless small performance improvements are worth largely increased training times. To maximize the probability of finding good solutions, the stopping should occur rather late, when the generalization loss (the ratio of the current validation error and the smallest validation error so far) exceeds a certain threshold.

Reduce Learning Rate on Plateaus

Glorot and Bengio [48] worked on "Understanding the difficulty of training deep forward neural networks". One of their results was that especially deep networks with sigmoid activations in the first layers get stuck on plateaus during training [9]. On the other hand, models are sometimes unable to find the global minimum, because their learning rate is too high [97]. In such a case, the learning rate could be decreased. Since the training worked well until the model reached the area close to the minimum, this decreased learning rate should only be a situational adaptation. Thus, the idea is to multiply the learning rate with a reduction factor if the monitored loss function is not reduced over several epochs. There are two options, to continue training with

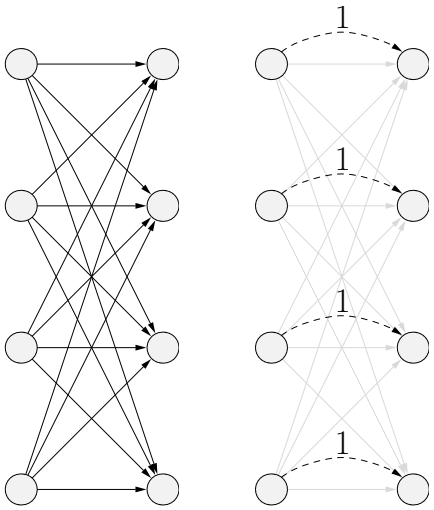


FIGURE 2.22: Dense layer with and without skip connection

the reduced learning rate, or to keep it only for a few epochs and return to the original learning rate. When assuming that the plateau was only an intermediate step on the way to the global minimum, I recommend to switch back to the original. Otherwise one can also keep the adaptation. In an experiment series performed by Maksutov [97], he finds the *best* result with applied reduced learning rate on plateaus. However, ranks 2 to 8 use models with a fixed learning rate. Therefore it seems that this is one of the options for which it is difficult to know in advance if it should be applied or not.

2.4.14 Residual Connections

The approach of *residual connections* was very successful for images and won 1st places in the ILSVRC and COCO 2015 competitions. The principle is motivated by the question: Is deeper always better? Consider a layered network N, to which one adds another hidden layer, which can learn weights to produce either the identity or some other function, thus creating the network N'. Note that passing the identity requires the same amount of neurons in two consecutive layers. Because each solution represented by N can also be represented by N', one expects N' to perform at least as well as N. While this is correct in theory, experiments show [60] that networks with an additional layer lead to worse results: Deeper is not always better.

Research on the exact reasons for this is still ongoing. One hypothesis is that the network struggles with learning the identity in the additional layer: Unfortunately, for optimal training, the weights need to be initialized around zero. Thus, the initialization is far away from the identity function and therefore the chances are small to actually reach it. So there is a dilemma: for the argument with the identity function some weights need to be close to one, while for optimal training weights are needed around zero. And — in retrospective — the solution seems obvious. One adds another link between the layers, a *skip connection*. It passes on the one value from the previous layer

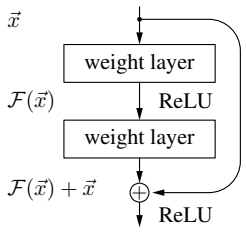


FIGURE 2.23: Residual Block, skipping two weight layers

and is added to the result of the original pass through the network. The skip connection is fixed and will not be affected by the training process and thus does not create additional trainable parameters. Further, it does not change the variance of the weights, and the computations to avoid vanishing and exploding gradients still hold. The original connection then only needs to learn the *residual* and is thus called *residual connection*. A simple form of skip connection is depicted in Figure 2.22 for a dense layer with four neurons. Each neuron in the earlier layer has an associated neuron in the subsequent layer which receives the additional input such that there are two connections between them, the normal connection and the skip connection.

An alternative to a skip connection would be to change the weight initialization such that the normal and the skip connection are merged back into one. At start of the training process, the computations of this variant are identical to the one with skip connections. However, the gradients would differ. This leads to a research question:

> **Q3:** *Is it sufficient to change the initialization to the identity function instead of using a skip connection?*

Changing the weight initialization can be done in different ways. In my experiments, I chose to keep the initialization as before using Glorot uniform initialization (see Section 2.4.8), but add the 1 values, where the skip connections would be. Since the implementations of layers are normally done by a matrix multiplication, it suffices to add the value 1 to each entry of the main diagonal of the weight matrix.

Variants of Residual Blocks

Residual connections are very successful when they are used to build so called *residual blocks*, which have identical numbers of input neurons and output neurons. The identity function can then easily be forwarded. A schematic view of a *residual block* is depicted in Figure 2.23. This processing block computes a mapping $\mathcal{F}(\vec{x})$, the skip connection (on the right) passes $\vec{x}$ unchanged to the output of the block, where the mapping and the unchanged input are summed so that the entire block computes $\mathcal{F}(\vec{x}) + \vec{x}$. Note that the skip connection does not change the set of possible outcomes of the residual block, but merely changes the default output.

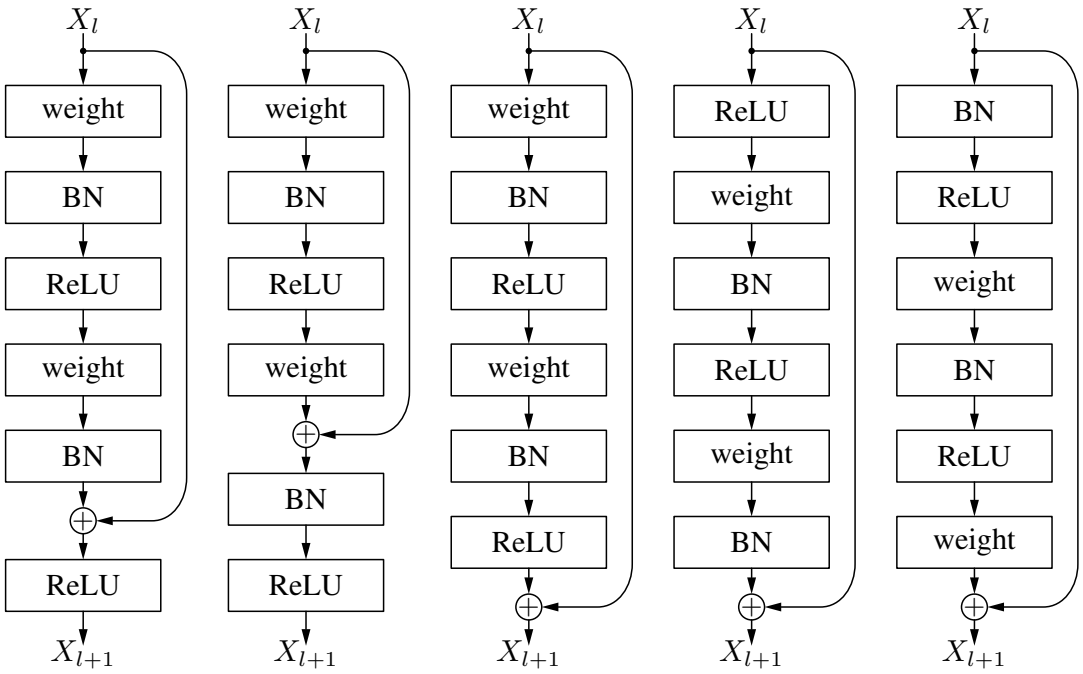


FIGURE 2.24: Different versions of residual blocks, described from left to right: (a) original, (b) BN after addition, (c) ReLU before addition, (d) ReLU-only pre-activation, (e) full pre-activation. Source: [60] (adapted)

The residual block from Figure 2.23 is very commonly used. However, I could not find answers to the following two questions:

1. Why does a skip connection skip exactly *two* weight layers?
2. Why is the second ReLU performed after the addition?

For the first question, I can only guess that using two layers gives the residual block more possibilities than one layer would. Therefore it gives the network the ability to learn more advanced features in a block with several layers, compared to one or two residual blocks that skip only one layer. Another explanation is that a block skipping two layers is the smallest block that would fulfill the requirements of the universal approximation theorem (Section 2.4), if e.g., tanh would be used as activation function instead of ReLU.

The second question is not answered but also questioned in many works since residual connections were originally introduced. He et al. [60] experiment with five versions of residual blocks. Their variants are depicted in Figure 2.24. The original version (leftmost) and BatchNormalization (BN) after addition (second from the left) do not pass the identity function to the next layer directly. The three right versions do this and only permute the order of the layers inside a block. The rightmost version, called *full pre-activation*, shows the best performance. The residual network with 164 layers reaches a classification error of 5.46% on the CIFAR-10 data set.

The residual block has changed the design of the neural network architecture. Earlier, only layers were be combined to build a computation graph, which seems to have changed, since nowadays residual blocks are combined

as well. Multiple residual blocks can easily be used in a row if they have the same size, but the performance gain of each additional block diminishes. Thus, blocks with different sizes are also concatenated. As default, residual connections would exist within blocks, but no connection preserves the input signal across several residual blocks. The latter is difficult, since two consecutive blocks may have incompatible input and output dimensions. However, it would be nice if the original signal could also be preserved across these blocks.

If the spacial dimensionality between blocks increases from n input neurons to $m > n$ output neurons, He et al. [59] propose a shortcut that performs the identity mapping from the input neurons to a subset of n output neurons and ignores the remaining output neurons.

When pictures are processed, they often use the same width and height. The number of filters (or channels) build the third dimension. In order to adapt the number of filters of the residual connection, as alternative to the identity function, a projection shortcut can be used that applies a 1×1 convolution with m filters to match the dimension of the output space. For me it is most plausible to use the identity function in the shortcut whenever possible. Further experiments by He et al. [60] verify that other shortcuts like gating or 1×1 convolutions indeed cause higher training errors.

If the output of the earlier block is bigger than the input of the next block, then Max-Pooling or strided 1×1 convolutions (see Section 2.4.6) are often applied. Max-Pooling always keeps the strongest signal, but it varies depending on the signal; strides ignore a part of the signal but always pass the same part of the input to the next layer. In practice both spacial dimensions of pictures are often reduced by a factor two between blocks, while at the same time the number of channels is doubled, such that in total the size of the signal is halved (see for example [58]).

Results in the well known ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [29] show the success of residual connections in practice. In this challenge, pictures from 1000 classes are given in a total of over one million images and the true class has to be detected. Results have continuously improved over the past years. In 2011, the winners used a model that was not based on neural networks and achieved an error rate of 25.7%. Since then, only models based on neural networks won this competition: In 2012, AlexNet [83] won the competition with an error rate of only 15.3%. The winner of 2015 [60] had introduced residual connections and achieved an error rate of 3.57%. More details on this challenge are given in Section 3.3.1.

Residual Connections for EEG Signals

For digital images, the data consist of two congenerous spatial dimensions (height and width), which is why 2D convolutions are applied. A 2D convolution with a filter size of 1x1, applied on a grayscale image, performs a

linear transformation, which is applied to all pixels. This adjusts the brightness of the image. Although this only adds very few weights to the model, these convolutions improve the performance.

I adapt this idea for EEG data: EEG data could also be represented in two dimensions, time and channel. While the order is clear for time, channels are measured on the scalp, which is why there is no natural linear ordering for them. Therefore, I apply 1D convolutions along the time axis in my models: If applied for each channel individually with a kernel of size one and one filter, this adjusts the amplitude of each channel, which could help to cope with noise, that influences the signal amplitude of a whole channel. If applied to all channels at the same time, the 1D convolution uses a two-dimensional filter and I use bigger kernel size to detect patterns that show relations between concurrent signals of different channels.

The idea of consecutive *residual* blocks with decreasing spatial size cannot be directly applied on EEG data, since it hardly makes sense to apply pooling on signals from different channels. One possibility is to use a pooling factor of 4 along the time axis and (similar as for pictures) double the number of filters. This would reduce the size of the signal between residual blocks by 2 as well for EEG data. Another possibility uses a pooling factor of 2 and double the number of filters and would therefore keep the signal size. I experimented with both versions and found the latter to be working better. Note that for a compression along the time axis, it is helpful to use an input signal with a length $t = 2^k$ that is a power of 2. This avoids errors due to rounding. My source code can be found in the model files located in `models/*`.

Schirrmeister [128] applies convolutional networks on EEG data and also recommends to apply a convolution for each channel individually. This leads to several research questions regarding 1D convolutions and skip connections:

> **Q4:** *Does enhancing convolutional blocks with skip connections improve the performance when performing EEG time series classification?*
>
> **Q5:** *Do channel-wise 1D convolutions at the beginning of a model improve their performance?*
>
> **Q6:** *Is it helpful to use skip connections in the channel-wise 1D convolutions?*

2.5 Aggregation

Aggregation functions were intensely studied in the last decade. Their purpose is to summarize information from several input values into one single representative value. Since I do not further investigate their mathematical properties, I refer the interested reader to the work of Grabisch [51] for an extensive theoretical overview.

I want to investigate the use of aggregation functions in a classification context. When an object is too big to be classified directly, I split it into several

sub-objects, and combine their classification results with the help of aggregation functions. One approach in this field is to learn the optimal aggregation function automatically. This builds a second level of learning on top of the initially learned classifiers and is therefore called *meta-learning* which has become its own field of research.

The *stacking* procedure tries to improve an overall estimator. First, one or more estimators are fitted on the training data. Then the final estimator uses not only the training data, but also the results of the other estimators as input. Dzeroski et al. raise the following question "Is Combining Classifiers with Stacking Better than Selecting the Best One?" [42]. Unfortunately for the current work, this method relies on different types of classifiers.

If the predictions are particularly *diverse*, that is, if they make their errors for different sample cases, it can be proven that the combined predictor performs at least as well as the worst single predictor. The work of Dietterich [30] explains several techniques to generate ensembles of classifiers.

However, in my case I use several classifiers of the same type and therefore the predictions vary only because of the different input snippets and different initialization. A meta-learner would only be able to learn patterns of the randomness within the process which would only lead to stronger over-fitting and reduce the overall prediction quality. So, in my scenario meta-learning will not be considered.

In an application, when measurements are evaluated online, like in a brain computer interface, the actual prediction quality can be improved by combining the models' predictions instead of treating them independently. As the predictions are based on possibly partly overlapping parts of the signal, the resulting predictions are not likely to be independent.

In my work, I apply the following six basic aggregation methods:

1. sum of probabilities: *psum*
2. sum of squared probabilities: *qsum*
3. product: *prod*
4. maximal probability: *pmax*
5. (discrete) majority vote: *dvote*
6. majority vote with abstention: *avote*

Before explaining details, I define the situation formally:
From one classification instance θ, n sub-objects $\theta_i, i = 1, \ldots, n$, which I also call *snippets* or *patches* are extracted. The model creates a prediction $\vec{p}(\theta_i)$ containing the probability for each class in K. Note that in most of the literature several classifiers p_i are used for the same data θ and therefore they use $p_i(\theta)$. Here, I do not vary the classifier p but the snippets θ_i used. Further, I use $p_k(\theta_i)$ to denote the probability of class k for snippet θ_i and $\vec{p}(\theta_i) = (p_1(\theta_i), \ldots, p_c(\theta_i))$ as the vector of probabilities for all c classes.

In a classification task, the class with the highest probability is predicted and the actual probabilities are ignored. Their distributions might contain insights, which might help with gaining optimal prediction performance. So the question arises how to aggregate them. This offers the opportunity to use these earlier ignored probabilities.

Sum of Probabilities

Formally, I first define the *linear opinion pool* as combined probability distribution similar to the work described by Clemen et al. [24]:

$$\vec{p}(\theta) = \sum_{i=1}^{n} w_i \vec{p}(\theta_i), \quad \text{where} \quad 0 \leq w_i \leq 1 \forall i \quad \text{and} \quad \sum_{i=1}^{n} w_i = 1$$

and then compute the predicted class c:

$$c(\vec{p}(\theta)) = \arg\max_{k \in K} p_k(\theta)$$

To make sure that $\vec{p}(\theta)$ is a probability distribution, the weights w_i need to be between 0 and 1 and they have to sum up to 1. Setting all weights to $1/n$ makes the weighting also invariant towards permutation. The formula can then be simplified to

$$c_{psum}(\vec{p}(\theta)) = \arg\max_{k \in K} \left(1/n \sum_{i=1}^{n} p_k(\theta_i) \right) = \arg\max_{k \in K} \left(\sum_{i=1}^{n} p_k(\theta_i) \right)$$

Because $1/n$ is constant, it does not change the maximal argument and the computations can be simplified: *sum* up all the probabilities for each class and predict the one with the highest value. Although this process does not generate proper probability distributions, it produces identical results to computing the arithmetic mean and predicting the class with the highest value, and avoids unnecessary computations.

Sum of Squared Probabilities

When using the *sum of probabilities*, predictions close to random guessing should not worsen predictions. But in principle it is possible that one prediction is sure about the class, and several minor random fluctuations cancel out this prediction and result in the wrong class predicted. So, I reduce the influence of predictions close to guessing, by down-weighting these predictions. Here, I square each probability. Those values are then summed before the class with highest value is predicted. So in fact, I use the *sum of squared*

probabilities, instead of the *sum of probabilities*.

$$c_{qsum}(\vec{p}(\theta)) = \arg\max_{k \in K} \left(\sum_{i=1}^{n} p_k(\theta_i)^2 \right)$$

Maximal Probability

The most extreme approach to minimizing the effect of near random guessing is to ignore those completely. This could be done by setting their weights all to 0 and taking only those samples, for which the maximal values for each class are reached. This method is called *max* aggregation.

$$c_{max}(\vec{p}(\theta)) = \arg\max_{k \in K} \left(\max_{i=1..n} p_k(\theta_i) \right)$$

Product

Another option to reduce the effect of predictions close to random guessing is to take the *product* of the predictions instead of the sum.

$$c_{prod}(\vec{p}(\theta)) = \arg\max_{k \in K} \left(\prod_{i=1}^{n} p_k(\theta_i) \right)$$

Of course, the resulting values for the classes decrease when more predictions are available, but as at the end it is only important which class has the highest value, this does not matter. The combined model predicts the class with the highest geometric mean. Note that in the implementation, I compute the sum of the logarithms instead. Therefore, the corresponding distribution is known as *logarithmic opinion pool* [24]. This returns identical results but avoids numerical instabilities that might occur due to repeated multiplication with values smaller than 1.

Discrete Majority Vote

One answer to the question how one should aggregate several opinions is implemented in elections, i.e. by using a *majority vote*. Each snippet votes for the one class with the highest probability. Then votes are counted and the class with the highest number of votes wins the election and will be predicted. Respecting only the highest value is known as *winner-takes-all* principle. It occurs twice in the process, which can be seen by the usage of the argmax function: First, the voter has to decide who gets the one vote, and who gets nothing (of his vote). Second, the person with the highest number of votes is elected, the others are not.

$$c_{dvote}(\vec{p}(\theta)) = \arg\max_{k \in K} \left\{ \sum_{i=1}^{n} \delta_{k,\arg\max_{c \in K}(p_c(\theta_i))} \right\}$$

In the formula, counting the votes is reflected by the *Kronecker delta*. 1 is counted if the class has the highest probability, 0 otherwise, for each of the snippets.

$$\delta_{i,j} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

In the literature, majority voting is also called *accuracy by agreement*. "If more networks agree on their classification, the chance that the classification is accurate increases" [8]. In their first approach Battiti et al. reject to perform a prediction if not all classifiers agree to predict the same class. They use up to six different classifiers for the same samples and show that the combined accuracy significantly increases. This process avoids the critical cases, where classifiers disagree. Thus increased accuracy is the behaviour I would expect, since with more classifiers, it can also be expected that more samples are rejected. In the extreme case nothing would be classified, and no error would be made. However, the number of rejected samples is omitted by the authors. In their second approach they gradually loosen the rejection criterion. They find that combined models with more individual predictors outperform those with less predictors. Further, the more they loosen their rejection criterion, the worse their prediction accuracy becomes.

However, rejecting snippets does not help in practice. In this work, I do not reject any given input and therefore predict the class with the highest number of votes. Here, a general problem with voting systems shows up: It can not be guaranteed that the overall aggregation results in a single class with most votes. Although for two classes this can be guaranteed by using an odd number of samples, this is not generally possible. In the extreme case, for every sample, the network predicts a probability of $1/n$ for each class. This can happen if the model gets stuck at a state, where $1/n$ is predicted independent of the input. This problem can be solved as follows: If two or more classes achieve the highest value, one of them is chosen randomly.

Majority Vote with Abstention

One issue with elections is that voters may be unhappy with the available possibilities and are therefore unsure what to vote. Two or more possibilities might be close to each other. In a real election, every voter has to decide for one possibility or abstain from voting. In this context, the winner is the one class with highest probability, while the other probabilities – although greater than zero – are ignored.

This can be implemented in many ways, for example the voter abstains if

a) the difference between the two most probable classes is smaller than a given threshold

b) the quotient of the first and second most probable class is smaller than a given threshold

c) the Shannon entropy of the class probabilities is above a given threshold

d) the Gini impurity of the class distribution is under a given threshold

Shannon entropy and Gini impurity are measures for statistical dispersion. High values of Shannon entropy indicate that the probabilities are close to random guessing since Shannon entropy is maximized by a uniform probability distribution. Unfortunately, this is not the exact problem of deciding which class to choose. The Shannon entropy of a probability vector $\vec{P}$ with n possible classes can be computed as $\mathrm{H(P)} = -\sum_{i=1}^{n} P_i log_n(P_i)$ [132]. Note that the basis of the logarithm in this definition is often 2 to obtain the unit of bits. However, I choose the definition with the base n such that the function is normalized to return 1 when the probabilities show random guessing and to return and 0 if the probabilities show that one class is certain. For three possible classes with probabilities (0.0, 0.5 and 0.5) the fact that class 1 is not possible is represented in the entropy, showing a value of approximately 0.631. In this example the classifier contains some information, but it is still impossible to decide between classes 2 and 3.

The Gini impurity is computed as $1 - \sum_{i=1}^{c} P_i^2$ [15]. It can be interpreted as the probability of a classifier being wrong, when no information but the relative frequencies of the classes are given by the vector $\vec{P}$ and the classifier should resemble the class frequencies perfectly in its predictions. Then, the probability of the classifier to correctly predict class i is P_i^2, since P_i is the probability for class i to occur, and P_i is also the probability of the classifier to choose i. The chance of the classifier to correctly predict an arbitrary class is thus $\sum_{i=1}^{c} P_i^2$. And the probability of the classifier being wrong is then $1 - \sum_{i=1}^{c} P_i^2$.

In the following experiments I chose variant *a*. This means that voting is abstained from, if the two most probable classes for a sample have probabilities that only differ slightly [8]. Still, the class with most votes is predicted, but the number of votes may be lower than the number of samples used.

For the implementation, the question arises how to define when probabilities differ only slightly. So the *vote with abstention* needs an additional parameter μ to model this threshold. The voting is abstained from iff the difference of the first and the second most probable class is smaller than μ.

$$\eta(v, \mu) = \begin{cases} 1, & \text{if } max_1(v) - max_2(v) \geq \mu, \\ 0, & \text{otherwise .} \end{cases}$$

For a simpler mathematical formulation, I use the function $max_k(v)$, which returns the k-th biggest value of the vector v.

$$c_{avote}(\vec{p}(\theta), \mu) = \arg\max_{k \in K} \left\{ \sum_{i=1}^{n} \eta(p_k(\theta_i), \mu) \cdot \delta_{k, \arg\max_{c \in K}(p_c(\theta_i))} \right\}$$

Implications on the Assumptions of the Snippet Distribution

One problem when multiple patches from a much larger object are given, is that the quality of each patch for the prediction task is *a priori* unknown. If I had to set weights on them, the only plausible choice would be to set them all equally, as this is the only option to achieve weights that are invariant with respect to permutation. Several questions arise:

> **Q7:** *Are there snippets, that are better suited than others? Can the class be seen in every snippet? What is the distribution of the snippet quality? Are there snippets that do not contain any helpful information? Are there snippets that mislead the classifier?*

All these questions ask for similar aspects of the snippets' distribution. For example it may be that *each* snippet contains information allowing conclusions towards the true class, or that only *some* snippets do. Additionally, some might contain more helpful information than others, or they may even contain misleading information. Depending on the assumptions on our data, I hypothesize that different aggregation methods also perform differently.

Chapter 3

Related Work

My own investigations on smokers in this work (in Chapter 5) are purely based on one data set. I apply

1. machine learning models,
2. using data from *human* brains,
3. measured by an EEG device,
4. at resting state,
5. involving smokers and non-smokers.

Therefore they can not give any insights on the mechanisms within the human brain, but give indicators for biologists and neuro-scientists, where they could intensify their research for mechanisms. Earlier research in neuroscience varied in one or more of these points. I start with research that varies in several of these points and approach more similar studies later on. I first give a few samples of more general research in the field of addiction, where some of the experiments were conducted on rats. This might be related since it is believed that several addictions are processed similarly by the brain. Similarities are also expected to be present between brains of rats and humans. Second, I present works related to EEG analysis: I start with resting state analysis, which aims for differences during rest, and continue with event related analysis, which investigates the brain's reactions to (e.g., acoustic or visual) events. At this point, I also add a few studies related to my own, which use data from functional magnetic resonance imaging (fMRI), instead of EEG. Finally, the last section refers to works which are not related to addiction, but show examples of areas where machine learning and especially deep learning were applied with great success. It further describes the differences in those tasks and highlights the emerging difficulties of EEG analysis.

3.1 Addiction

Addiction is a major field of research. In order to illustrate the multitude of research directions I present a few examples, like reaction tests performed on

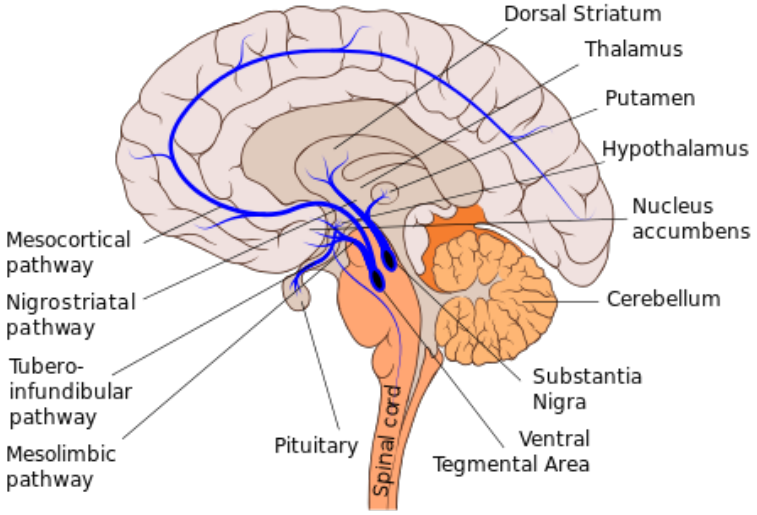


FIGURE 3.1: Brain region and dopaminergic pathway schema (Source: [66])

rats, the analysis of the level of the neurotransmitter dopamine, and comparisons of brain activity in people addicted to food or alcohol. The following summary is intentionally limited to the conclusions since I want to keep the focus on what is relevant to my work, instead of delving deep into neuroscience, which is also not my field of expertise.

A meta-study performed by Jasinska et al. [76] investigated "Factors modulating neural reactivity to drug cues in addiction". They summarize the current research of the field and separate two main groups of influence factors being studied: study-specific factors and individual-specific factors. To the first group they assign drug availability, sensory modality and length of presentation of drug cues as well as explicit and implicit regulation. The stronger factors of the individual-specific patterns are the addiction severity and the current treatment status — is the person an active user, trying to quit, or abstinent — and the weaker factors are the length and intensity of use, the length of abstinence and severity of withdrawal and the 'stressor exposure', defining if the drug cue was recent, chronic or at early age. All of these factors influence the neural reactivity to drug cues. They list the following brain regions to be influenced: the mesolymbic system, the mesocortical system, the nigrostriatal system and other brain regions like the cerebellum, from which many are related to dopamine. A schematic is given in Figure 3.1. It shows the so called *dopaminergic pathways* which are sets of neurons that synthesize and release the neurotransmitter dopamine.

Dopamine is involved when neurons send signals to other neurons, and addictions are known to influence the level of dopamine. Noble [106] summarizes the research concerning a specific dopamine receptor gene, the *DRD2* and its variants. This gene is linked to nicotine-dependency, alcoholism, cocaine-dependency, opioid dependency, obesity and gambling.

De Ridder [28] investigated alcohol addicted people, obese people with and without food addiction and non addicted lean controls. Food addiction is

measured by the Yale Food Addiction Scale [45]. They found indicators for a "common substrate hypothesis" for food and drug cravings, meaning that those addictions are related to similar active areas within the brain. For alcohol addicted and food addicted people, these similar activities were found in several brain areas. They also found that "food-addicted differ from non-food-addicted obese people by opposite activity" [28] in one brain area: the anterior cingulate gyrus.

For rats Pich has shown [116] that both, the self-administration of nicotine or cocaine have similar effects: High levels of the *AP-1-protein* complexes were observed in the nucleus accumbens. These complexes are important for the conversion of DNA to RNA (the transcriptional regulation). This strengthens the indication of a common neuronal substrate for addiction to cocaine and nicotine.

Summarizing, similar brain regions are related to different addictions, from substance addictions like alcoholism to gambling. Many studies find relations to the dopaminergic system, which controls the distribution of dopamine within the brain. Unfortunately, these pathways span multiple brain areas, such that no clear conclusion can be drawn on which location of the scalp (or more specific, which channel) should be especially expedient, when investigating addictions with the help of EEG signals.

Q8: *Which brain areas are best suited for the prediction of a given task?*

3.2 Brain Data Analysis

Research using EEG has typically looked at patterns in various frequency bands to determine if there are differences between groups of participants. Many studies investigate differences in the signals while patients were at resting state, in contrast to studies where the *reactions* of subjects on external signals are investigated.

3.2.1 Resting State Analysis

EEG Epilepsy Analysis

One of the typical applications of EEG is in the diagnosis of epilepsy. For example one study was performed by Omerhodzic et al. [108]. They used data from three groups: healthy subjects, people during an epileptic seizure and people diagnosed with epilepsy, but not having an acute seizure. For their comparison, they used a wavelet transformation to split the signal into six frequency bands and used the detail and approximation coefficients values of each band as features for their neural network classifier. Unfortunately, the computations of these features is missing. However, a visualization of

the energy levels within the bands is presented, which already shows visually that there are significant differences within these features: A low energy in the gamma band seems to be characteristic for epileptic people when having no seizure, and higher energy levels in the beta, alpha, or theta bands are characteristic for an epileptic seizure. Consequently, a neural network classifier with only one hidden layer with 5 neurons was able to perform the classifications with a relatively high accuracy of 94%. I hypothesize that if those features were extracted for each channel individually and not directly aggregated, that the classification accuracy could have been even better.

This is just one example that shows the success of EEG analysis. However, it also shows that the problem becomes rather simple, once a preprocessing is known which extracts the important features. Once the important features are known, they can be analyzed to build theories on neural processes, and they can later be used during the preprocessing, to enable applications with better interpretable models. However, this assumes that it is *possible* to extract the important features, which needs to be done individually for each task.

EEG Smoking Analysis

For a resting state analysis of smokers, the findings are rather inconclusive, although they do point to the potential for characteristic differences: For example, Brown [17] found reduced alpha and increased rhythmic high frequency when looking at the EEG signals of smokers vs. non-smokers. Rass detected reduced alpha as well, but also observed reduced delta when comparing smokers to non-smokers [120]. Additionally, Knott [81] reports reduced delta and increased beta within smokers as a function of image-induced craving. Together, these studies suggest that there may be differences present, but the definition of the precise patterns that lead to such differences needs further exploration.

fMRI Smoking Analysis

Previous research on addiction has generally used functional magnetic resonance imaging (fMRI) to examine differences at the level of resting-state data in smokers. It measures the blood-oxygen-level dependent (BOLD) within the brain by detecting the magnetic differences of oxygen-rich blood and oxygen-depleted blood. It is used as an indirect indicator of activity based on the knowledge that active neurons consume more oxygen. fMRI is an expensive method, due to the relatively large and expensive device needed for the measurement. It yields relatively low temporal resolution and restrictions in subject populations since it demands a metal-free environment. This excludes people with metal implants, piercings, and even people that have tattoos with colors that contain metal, since they would cause burn injuries during the measurement.

Differences have been found between subjects who are craving or sated [90], or between smokers and non-smokers [140][151], with frontal, executive-control-related regions such as the insula or dorsolateral prefrontal cortex (DLPFC) being implicated in those differences. Previous modeling techniques by Pariyadath et al. in 2014 used machine learning to determine smoking status in fMRI data [109] and applied support vector machine (SVM) classifiers and reached at best 76.2% accuracy on a data set consisting of 21 smokers and 21 controls.

In 2019, Wetherill et al. [152] used a data set consisting of 108 people who were not only smokers, but were diagnosed with nicotine use disorder and 108 people from a control group. Similar to the work of Pariyadath et al. [109], they also used a support vector machine classifier with a very extensive testing method: They repeated their experiments 10 times with different random seeds. In each repetition they performed a 10 fold cross-validation, where the testing set used the data from one fold and training was performed on the other nine folds. And within each of the 10×10 runs, they used another 10 fold cross-validation for their feature selection. Finally, their classifiers reached an average accuracy of 88.1%.

3.2.2 Event Related Analysis

One more traditional way to analyze EEG data is to conduct an event-related analysis. In this form, a subject is given a task, and every time an event is presented (e.g., a picture or a sound), a code signal is added to the data such that the reactions can be easily analyzed after a stimulus. For example the *P300* [141] (also called *P3*) is a reaction visible in the EEG signal after 250 milliseconds and 500 milliseconds. A subject is shown a picture consisting of several items of which one is to be freely selected by the subject. Then the selected item can be found by highlighting the items randomly, because the P300 reaction differs for the chosen item. This interesting behavior has been widely used since its discovery. Generally, event related data can be used for time-locked selective averaging.

In 2010 Versace et al. [147] found that the P300 reaction to smoking-related pictures was similar to the reaction to pleasant and unpleasant pictures. They saw a significantly reduced P300 component compared to the reactions to neutral pictures. They conclude that cigarette-related images as well as pleasant and unpleasant images "capture attentional resources and reduce the smoker's ability to process other competing stimuli" [147].

In a following study from 2011, Versace et al. [146] investigated two other effects: First, late positive potential (LPP) is recorded after 400 to 700 milliseconds at the central and parietal sites. Similar to the P300, it was found to be stronger when emotional or smoking-related images were shown. Second, the *P1* component is a factor already visible after approximately 140 milliseconds. It was found to be similar for emotional and non-emotional images, but it showed enhanced positivity for smoking-related images.

Donohue et al. [40] identified different reactions to pictures when subjects had the desire to quit smoking, compared to a second group which did not want to quit. "...smokers wanting to quit showed an enhanced late frontal activation when they were craving vs. not craving, whereas smokers not wanting to quit showed the opposite pattern of activity" [40]. Also, a similarly differing brain reaction showed up in the target-related P300 — again, opposite patterns occurred for those subjects that wanted to quit compared to those who did not.

In another work by Donohue et al. [39], they considered another effect visible in EEG data, the *N2pc* that is normally associated with a persons attention. When smokers were craving, they showed generally more arousal, subjects showed a larger P1 in their neural activity in response to all stimuli presented, and regardless of whether the shown image was nicotine-related or non-nicotine related. A second finding was that smokers directed their attention *away* from the smoking-related images (i.e., avoided them) regardless of their state of craving.

Note that the data used in [39] uses a partially overlapping population with data used in Chapter 5. While they used only data from smokers, and considered their reactions when showing them images, after these experiments the resting state signal was taken, which is analyzed in this work. They describe that their experiments with craving subjects started 3 hours after the subject's arrival and that the preparation of the cap took approximately 45 minutes. Since the reaction experiments were conducted before the resting state signal, it can be stated that in my own experiments, *craving* subjects had not smoked for at least 4 hours. It is an open question, however, if a significant difference as for the event-related study would be present in resting-state data as well, and if those differences are large enough to be captured by a machine learning algorithm.

A meta-analysis on reactions of smokers was performed by Grabski et al. [52] in 2016. Interested readers are referred to their work. They included 42 studies on acute abstinence as well as 13 studies on cessation in their analysis. Because of methodological inconsistencies, they could not make conclusions on the studies on cessation. However, comparing satiated with abstinent smokers revealed an impaired arithmetic and recognition memory performance as well as lower response inhibition for acutely abstinent smokers.

Event related potentials are a powerful means when analyzing neural data. For machine learning tasks, it seems natural to create one training sample per event, which does not only allow to use different events in order to evoke different reactions, but also to measure the reaction time, that may also be an helpful indicator. Resting state data does however *not* offer natural beginnings and endings of the evaluation of the measured time series. Since also no *reactions* can be measured, I assume this to be the more difficult task.

FIGURE 3.2: Eigenfaces (Source: [67])

3.3 Neural Networks as Feature Detectors

The following section is meant to give a few examples of applications of neural networks. In many areas neural networks are successful, even if earlier machine learning approaches may have failed. This underlines the difficulty of the tasks, and the huge advantage of a neural networks' ability to detect the relevant features for the given problem on its own.

3.3.1 Object Recognition in Pictures

One task that is fairly easy for humans is to detect objects in images. However, detecting objects in images is difficult for computers. Before neural networks were used for this task, the state of the art in 1991 were so called *eigenfaces* [145], which were prototypes of human faces that were used for face recognition. I show a few examples in Figure 3.2. Note that at that time, the task was to correctly *detect* a human face.

Very active research in that field and also competitions like the Face Recognition Vendor Test [55] favored the great success of current applications at identifying persons. The primary data set for the contest contains 26.6 million images of 12.3 million individuals. "With good quality portrait photos..." algorithms achieve "rank one miss rates approaching0.1%" [55].

For recognizing not faces, but different kinds of objects in images, the yearly ImageNet Large Scale Visual Recognition Challenge [29] shows the state of the art. The organizers offer a dataset with 1000 classes consisting of more than 1 million images.

In 2011, before neural networks were used, the top-5 error rate[1] of the best model was 25.7%. Since 2012 all winning approaches used neural networks. The winner of 2012, AlexNet [83], was able to reduce the top-5 error rate to 15.3%. In 2015, He et al. [58] used residual connections and thus were able to use much bigger networks than before. Their winning model used an ensemble of neural networks, where the biggest network consisted of more

[1]The top-5 error rate measures the ratio of incorrect predictions, which counts an error for a sample only if a set of 5 predictions does not contain the true class.

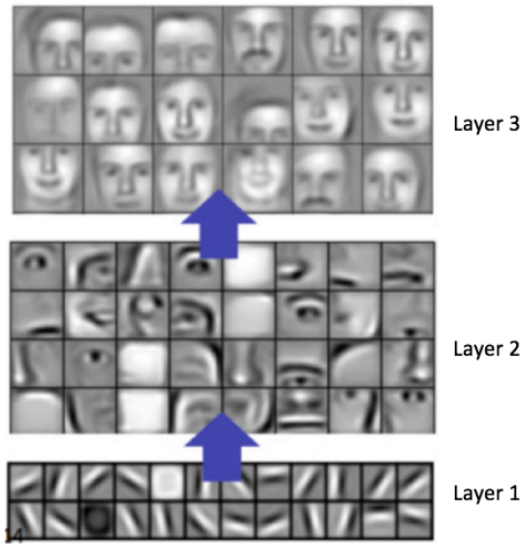


FIGURE 3.3: Visualization of several convolutional layers of a network trained to detect faces (Source: [89])

than 100 layers. The model yielded a top-5 error rate of 3.57% which was the first result that was better than the average human, who is expected to make errors at a rate of approximately 5%.

Visualization of Features

A nice property of images is that they can be easily interpreted by humans. I used visualizations of convolutional filters when I explained them in Section 2.4.5. In a similar way to predefined convolutional filters, trained filters can also be visualized.

Such a visualization is shown in Figure 3.3. In the first convolutional layer the trained filters detect edges in arbitrary directions and areas with very high and very low brightness. These are linearly combined to build the features from the second layer. The visualization of this layer shows features which look like parts of faces: eyes, eyebrows, noses, mouths. As before, these features are linearly combined to create a third layer of features. The visualized features look like faces. The details are mostly not visible, but the number of recognizable details is much higher compared to the eigenfaces from Figure 3.2. This visualization is a great tool to make sure that the network trained what was actually expected. However, in this example, it relies on the fact that humans can interpret pictures and can thus also interpret visualizations of features.

Localization of Objects in Pictures

After the great successes of face recognition and object recognition, more difficult tasks were considered: For example extending the problem *which* object

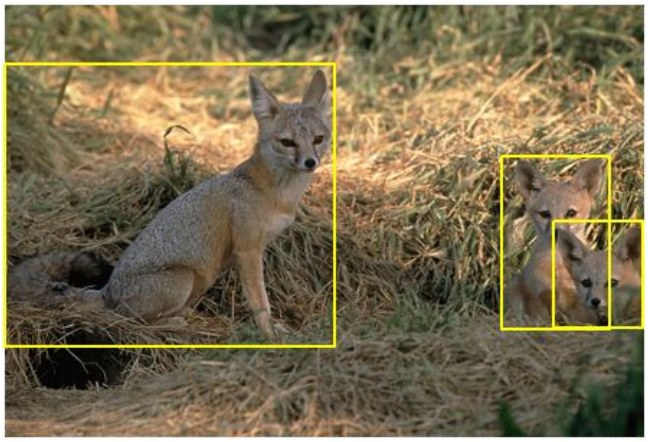


FIGURE 3.4: Foxes in the wild to be localized by algorithms (Source: [72])

is visible on a picture to the question *where* it is. The corresponding competition is hosted at kaggle [72]. This task is more difficult, also for humans, as the example of Figure 3.4 illustrates: Since the foxes are camouflaged in the wild, it is difficult to see where the fox ends and grass begins.

3.3.2 Finding Relevant Substructures in Molecules

The following example can be seen as a practical task from pharmaceutics. When new active components of medications are developed, a screening process is applied in which different candidates are investigated. Therefore one is interested in molecules that effectively affect a target in a specified way. Since there are huge amounts of candidates, a detailed investigation by a *full screening* of all of them is inefficient and thus costly. To avoid unnecessary costs, the screening process is divided into three steps: First, a few molecules are screened; second, based on this generated data, a *Virtual High-Throughput Screening* is applied, and finally the most promising candidates are tested. In the typical virtual screening process, the molecule structures are given and with the help of expert knowledge transformed in features vectors. These feature vectors are used as inputs for classifiers that perform predictions on the effectivity of the candidates. For those candidates that are expected to be most effective, a screening is applied. In the approach of Winter [153] he replaces the feature generator that uses expert knowledge with an automatic feature generation for which he uses neural networks (depicted in Figure 3.5).

Therefore he utilizes the improvements of neural networks applied to images: He renders images of molecules and then applies networks that are similar to the ones applied to find objects in images. In addition to using the neural networks directly as classifier, he uses a structure that allows him to split the trained network into the first part, which creates the features, and into the second part, which performs the classification. In order to compare the feature vectors of the experts with the ones the network generated au-

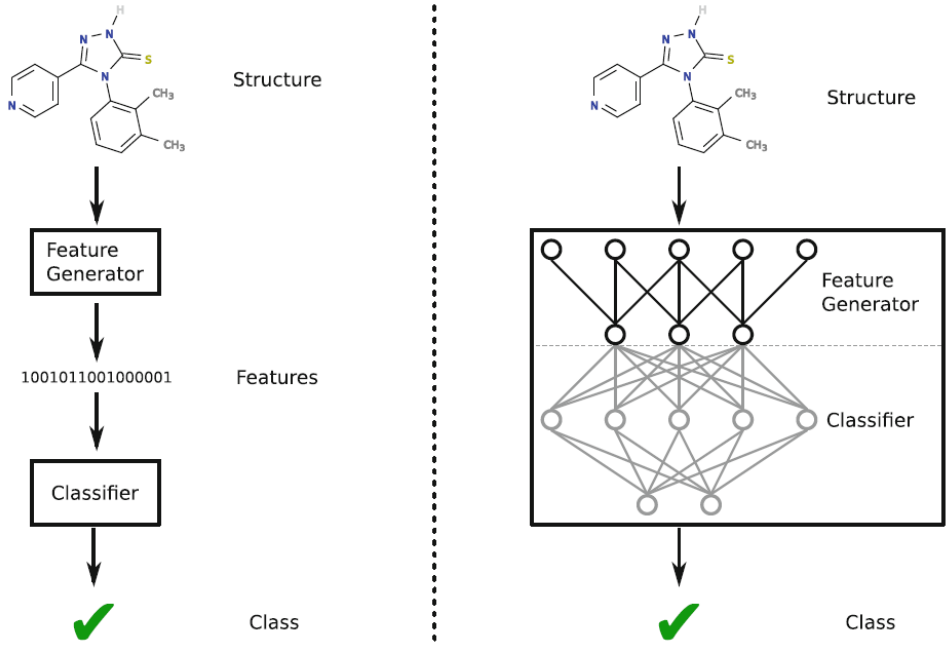


FIGURE 3.5: Detecting substructures of molecules: left with the classical method generating a feature vector with expert knowledge, right with the proposed approach to automatically generate the features data-driven using a neural network (Source: [154])

tomatically, he splits the network and uses the first part to create a data set consisting of the features learned by the network.

He finally uses this data set to train different classifiers for the same task. There is only a small difference of the prediction quality for other classifiers. Thus, the main result is that an automatic feature generation by neural networks is possible for this task and automatically generates similarly performing feature vectors without the help of human experts. These have the advantageous property to be interpretable for example as highlighted parts of a picture that shows the molecule.

3.3.3 Playing Games with Complete Information

Computer programs are best known to play games with complete information. This means games, in which all players have all information about the game situation. Chess allows roughly 10^{43} different positions [131]. This number demonstrates that solving this by computing all possible positions is practically impossible. However, it may be possible to defeat the best human. In 1997, this was shown by the chess computer *DeepBlue*, that defeated the world chess champion at the time, Gary Kasparov. This was achieved without the help of neural networks, but with the usage of massively parallel search and a computer that was especially built for this task.

Since in the game of Go approximately 10^{170} positions are possible, it was believed that it would take many years until computers would be able to solve it. However, one of the great recent successes of neural networks is the defeat of the best humans in the game of Go. Less than 20 years after the defeat of Kasparov, in October 2015 AlphaGo [134] defeated Fan Hui, the champion of the European Go Championship in 2013, 2014, 2015 and 2016 (the latter after his defeat against AlphaGo). In March 2016 it beat Lee Sedol, a 9-dan professional Go player, who was #1 on the world ranking list from 2007 to 2011 and #4 at the time of the match.

One part of the solution was of course the development of faster hardware. However, another part was better algorithms. AlphaGo combines two neural networks with the Monte Carlo tree search algorithm. One neural network called *value network* is trained to evaluate positions, and the *policy* network is used to sample game actions. The policy network is first trained by using games played by human experts. MonteCarlo tree search is a strategy that searches probable paths in the tree of possible actions of both players. In combination with the value network to evaluate game positions, the action (and the corresponding sub-tree) is chosen, which yields the best outcome, even if the opponent chooses his best moves. So called *rollouts* are randomly performed, in which simulations of the game are played to the end such that the outcome of the simulation can be used to adapt and improve both of the networks[2]. Once this architecture is set up, it can be improved by repeatedly playing against itself.

In 2017, the team of AlphaGo published a new and even better version, *AlphaGo Zero* [135], which was created without using the data from human games. By repeatedly playing against itself it exceeded all previous versions of itself within 40 days of training.

3.3.4 Comparison to EEG Data

In addition to the examples from above, there are many other successful applications of deep learning on biological data [96], like medical imaging or brain computer interfaces. Having the successes from above in mind, I want to emphasize the difficulties to classify data from biological sources, using EEG data as an example, by comparing it to the above tasks.

Number of Data Samples

In the tasks described in the preceding sections, plenty of data is available. The Imagenet challenge uses a database of millions of images, in which objects are to be found. Mastering the game of Go can rely on the current model

[2]Note that the training procedure applied here is not the earlier explained backpropagation but so called *reinforcement learning*. Since I do not apply reinforcement learning in this work, I refer the interested reader to the work of Kaelbling et al. [78].

playing against itself in order to generate more training data. Therefore, more training data can be constructed whenever needed and the number of games is, in principle, unlimited.

In stark contrast to the favorable situation in these two tasks, measurements of EEG data are very limited. There are several reasons for the small number of participants:

1. Taking measurements is expensive.
2. It is difficult to find reliable participants since they often only get a small financial compensation.
3. There are many legal restrictions. Especially since the data is individual-related, many are not publicly available.
4. Due to the individual-related data, the sampling process needs to handle this by using a procedure that respects the individual-related groups of data samples (explained in Section 2.3.2).

Standardized Data

In the above tasks, all data is standardized. For example if one data point was lost, it would be possible to replace it with another one: A molecule can be screened again with high chances of a similar result. A picture can be taken again. But it would probably differ in some aspects: Even if the same photographer took another picture with the same camera of the same motif, since some time has passed between taking the original and the replacement, the pictures would differ. But they would also show significant and recognizable similarities. If the latest version of AlphaZero was lost, the documentation is available, which would allow to use the last available version and compute a similar version again.

For EEG data, another measurement could be taken, but it takes much effort to get one as similar to the original as possible. The same EEG device, and montage should be used. The electrodes should be placed just as they were originally placed, best by the same person. And finally, similar to pictures, the motif (the measured subject) has changed during the passed time. And afterwards identical preprocessing steps should be applied. Still, this does not fully capture the difference: Facial features like shape, or eye color are more stable than the processing patterns in the brain, simply because the person would not think about the same things or in the same way. This should make an EEG much more volatile. Thus I conclude that it would be difficult to construct an EEG measurement similar to the original.

Noise in the Data

EEG measurements are prone to noise, from several biological and technological sources that can not always be avoided (see Section 2.1).

For the game of Go, there is no noise in the data. The possible positions of the stones on the board are fixed. Even if an error occurs when a move is recorded, this can be fixed in many cases by the information contained in the other moves. Noise in pictures is different and can depend on the used camera, the lighting at the moment when the picture was taken, or the aptitude of the photographer. Still, these problems can (usually) be solved easily by removing bad or noisy images, since humans have the ability to compare pictures to reality and it is cheap to simply take another picture if it shows blur that might be caused by camera shake.

Knowledge of the Task and of Important Features

Most of the people have the ability to recognize objects in pictures. Furthermore, they can explain why they think the picture contains an object: They can describe where they see legs, arms, the head, or other features, that might help with the classification of the object in the picture. Therefore humans can compare their interpretations of images. This understanding is so robust that often neither modifications of the image in brightness, scale, rotation nor the fact that the object of interest is partially covered, prevents humans to recognize it correctly. Even the first part is not true for neural networks: slight modifications may lead to a wrong prediction, as demonstrated, for example, by so called *adversarial examples* [50].

For the game of Go, a vast body of expert knowledge exists, so that it can be validated whether a model plays moves similar to those a human expert would play. If the moves are different, but the model still wins, experts can even gain new insights into the game and how to play it well, which is what happened for Go. Even more importantly, experts can easily detect *bad* models, by playing against them, and winning.

When analyzing data from neurological sources, with the research goal to *gain new* insights, then only limited knowledge is available that could be used to verify the correctness of the results or the relevance of the found features.

Concluding Remarks

In the last sections, I explained the particular differences that biological time series data bring, and compared them to tasks with data from other sources, where neural networks have been successfully applied. The most important difference is that the measurements are individual-specific. Thus, when interested in models that are able to generalize to other people, these models need to be trained on data from many different subjects.

The individual-specific structure of the data also leads to particular difficulties when the data is split into training set, validation set and test set. Thus I conclude that studies that rely on biological time series data should be treated

with special attention. For these reasons, I am especially suspicious if models present a combination of new preprocessing methods and new, complex network architectures that are specifically tailored to one task, e.g. classifying the state of schizophrenia [157] or differentiate between different seizure disorders [4]. There is a possibility that the models are overfit to the given data set.

In order to build reliable results, I recommend to perform replication studies if possible. Another means to reliability is to create models that can easily be interpreted, such that it is possible to gain insights into the problem by analyzing the found models. Unfortunately, biological time series are difficult to interpret, since there is no format known that makes them easily interpretable to humans.

For the field of smokers and addiction, I try to achieve interpretable results by using interpretable models first. Since they show poor performance, I try neural network models, which promise better results at the price of worse interpretability. Finally, the resulting model is presented to an expert, with the goal to understand it.

Chapter 4

Aggregation on Simulated Data

This chapter takes its basis from my own previously partially published work "Aggregation of Subclassifications: Methods, Tools and Experiments" [34] and extends it: I explain the sampling process with additional pictures, give results of experiments with a higher parameter range for the Dirichlet bias, and finally give a wider overview of the results.

A problem frequently encountered during the practical process of classifying complex objects (e.g., images, time series, graphs) is the difficulty of treating them directly. One approach to solving this problem is to split these complex objects into several smaller sub-objects (e.g., snippets of time series or patches of images) and classify them separately before aggregating the (possibly probabilistic) subclassifications.

Although aggregation functions also have a more general significance (see Grabisch et al. [51] for an extensive theoretical overview), my focus here lies on the question: Which specific functions work best to aggregate such subclassifications? A common approach to aggregating subclassifications is to learn an aggregation function. This adds a second level of learning on top of initially learned classifiers and is therefore called *meta-learning* or more specifically *stacking* [155], which has become a field of research in itself. An important question was raised by Dzeroski et al. [42]: "Is Combining Classifiers with Stacking Better than Selecting the Best One?". However, the suggested method is geared toward combining different types of classifiers that are applied to the same object. In contrast, I consider the case of a single classifier that is applied to different sub-objects (snippets, patches). Although some fundamental issues remain the same, in this case a meta-learner might only be able to learn patterns of the (random) sub-object selection process and thus might lead only to stronger over-fitting and hence reduced overall prediction quality. Therefore I do not consider meta-learning here.

The approach I will focus on is known as *mixture of experts*, in which an optimal (but simple) rule is sought to aggregate classifications (resulting either from different classifiers or from different sub-objects) into a single classification [156]. In this case it is usually assumed that each subclassification is a probability distribution over the classes, thus allowing for rules other than a simple voting scheme. Xu and Amari define a general class of such combination rules, which are known as f-means [156]. Of these, several of the rules

I consider in this work are special cases, for example forming the sum and forming the product of the probabilities per class. However, I also investigate different voting schemes and a few combinations of basic aggregation rules, as described in Section 2.5.

A core problem of the simulation approach is the modeling of biases. In any classification problem it is assumed that there is a bias towards the true class. Otherwise predictions better than random guessing cannot be expected with any classification algorithm.

4.1 Generating Finite Probability Distributions

4.1.1 Creating Unbiased Data

When modeling bias, a simple approach is to create unbiased data first, and introduce bias afterwards in order to investigate the effect of different types of bias and different strengths of biases.

In a classification with k classes, the classifier outputs a probability value p_k for each class, and respects the property of probabilities:

$$\sum_{i=1,\ldots,k} p_i = 1$$

The straightforward idea to generate a finite probability distribution would be to draw k samples from the uniform distribution and then renormalize them to sum up to 1. Unfortunately, this method leads to a bias. I visualize the effect in Figure 4.1 for $k = 3$, where samples are drawn from a cube in three dimensions. The normalization is equivalent to a projection onto the triangle that represents the plane for which $p_1 + p_2 + p_3 = 1$. A sampled point $s = (s_x, s_y, s_z)$ is projected to $p_1 = 1$ if and only if $s_y = s_z = 0$, while s_x may be an arbitrary value in the interval $[0, 1]$. The probability of this happening is proportional to the length of the corresponding line. In this case, the length is 1. In contrast, if s is projected to $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ it can be deduced that the original point was on the cube's diagonal which has length $\sqrt{3}$ and thus is more probable. This effect generates a bias towards the center of the triangle. The right part of Figure 4.1 depicts 1000 points drawn and projected to the triangle. The points lie more densely towards the center of the triangle. For a hypercube in n dimensions, the effect becomes stronger for increasing numbers of n because the length of the main diagonal is $\sqrt{n}$. To receive unbiased data with a similar approach, one needs to draw points not from a cube but from a sphere.

The canonical way to obtain a finite probability distribution in probability theory is to sample from a Dirichlet distribution [107, 82]. It is usually written

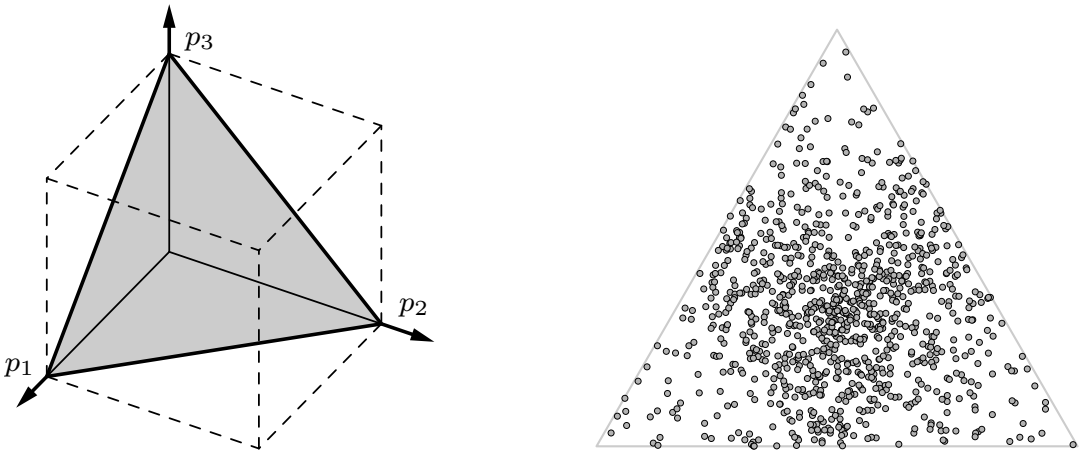


FIGURE 4.1: After sampling points from a unit cube, normalization means projecting them on the plane shown as a triangle, where $p_1 + p_2 + p_3 = 1$. Left: The triangle in the 3D cube. Right: 1000 sampled and projected data points.

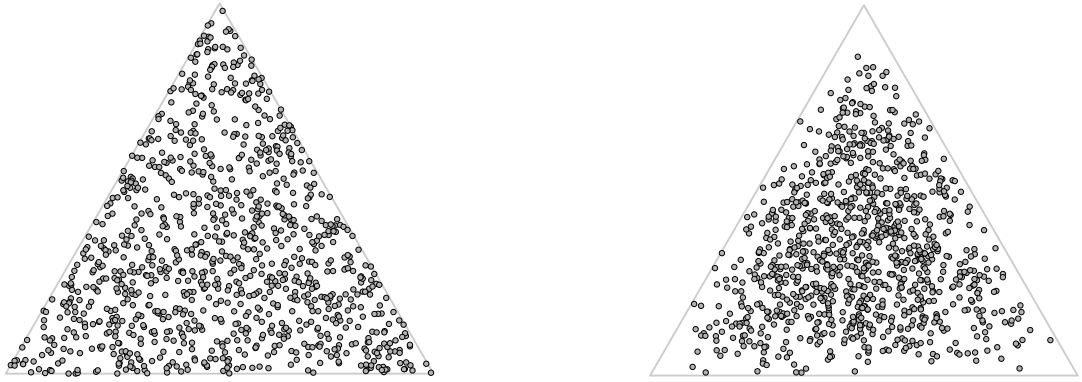


FIGURE 4.2: Sampled 1000 data points, left from $Dir(1,1,1)$, right from $Dir(2,2,2)$

as $Dir(\vec{\alpha})$ and uses the density

$$f(\vec{x};\vec{\alpha}) = \frac{1}{B(\vec{\alpha})}\prod_{i=1}^{k} x_i^{\alpha_i - 1} \quad \text{with} \quad B(\vec{\alpha}) = \frac{\prod_{i=1}^{k}\Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{k}\alpha_i\right)}.$$

Here $\vec{x} = (x_1, \ldots, x_k)$ is a vector in the $k-1$ simplex, that is, $\forall i, 1 \leq i \leq k$: $0 \leq x_i \leq 1$ and $\sum_{i=1}^{k} x_i = 1$, and the vector $\vec{\alpha} = (\alpha_1, \ldots, \alpha_k)$ contains a set of (hyper-)parameters. The normalizing constant $B(\vec{\alpha})$ is a multivariate beta function, expressed in terms of the Gamma function,

$$\begin{array}{rcll} \Gamma(n) & = & (n-1)! & \text{for} \quad n \in \mathbb{N}, \\ \Gamma(z) & = & \int_0^\infty \alpha^{z-1} e^{-x} dx & \text{for} \quad z \in \mathbb{R}^+. \end{array}$$

A natural choice is $\mathrm{Dir}(\vec{\alpha}_r)$ with $\vec{\alpha}_r = (r, r, \ldots, r)$, as this treats all classes equally. This is also referred to as the symmetric Dirichlet distribution. The special case $\vec{\alpha}_1 = (1, 1, \ldots, 1)$ yields a uniform density on the $k-1$ simplex, namely $f(\vec{x};\vec{\alpha}_1) = 1/(k-1)!$. With this choice, every probability distribution

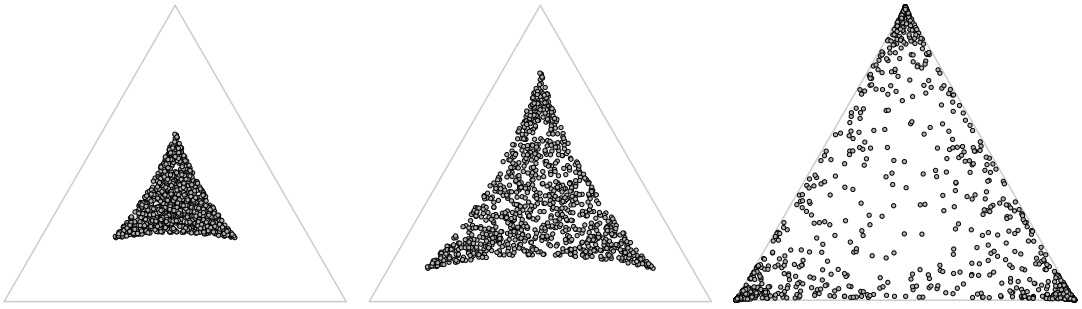


FIGURE 4.3: Examples of distributions when points are drawn from Dir(1,1,1), then scaled by factor 1,2 and 8 (from left to right) and afterwards applied softmax function

over k classes is equally likely. For $r > 1$ a bias toward a uniform distribution with a strength quantified by $r - 1$ is introduced, that is, points in the center of the $k - 1$ simplex have a higher density than those near its corners as shown in Figure 4.2. This effect could be interpreted as having fewer extreme values, or fewer outliers within the data.

In an application that applies a neural network and a final Softmax layer (Section 2.4.3), I assume that the opposite effect occurs — many points would be close to the triangle's corners. Figure 4.3 visualizes the effect of the softmax function, when it is applied to data sampled from a Dirichlet distribution $(\text{Dir}(\vec{\alpha}_1))$. Left, it is applied directly, meaning that the input range is limited to the interval $[0, 1]$. But the assumption that the input is so strictly limited does not hold in the practice of neural networks. In fact, one reason why softmax is applied so frequently as final layer is that it maps an arbitrary input from $[-\infty, \infty] \to [0, 1]$. For the middle and right plot of Figure 4.3, the data stem from a scaled version of a Dirichlet distribution $(\text{Dir}(\vec{\alpha}_1) - (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})) \cdot s$, with s as scaling factor — $s = 2$ in the middle and $s = 8$ on the right. For small scaling factors softmax seems to 'shrink' the space. This is plausible, for example $(1.0, 0.0, 0.0) \mapsto (\frac{e}{e+2}, \frac{1}{e+2}, \frac{1}{e+2}) \approx (.576, .212, .212)$. However, for the practice the question is still open:

> **Q9:** *Is there a bias towards the corners of the k-class projection simplex, when applying a neural network with a softmax layer to real data?*

Note that the I do not create a measure, that would allow to answer this question for arbitrary values of k. Instead, I focus on $k = 3$ for the practical reason that the resulting triangle can be easily visualized to answer the question.

4.1.2 Introducing Bias

To date, I have only considered how to generate finite probability distributions on k classes that treat all classes equally. This is appropriate for those sub-objects that do not provide information about the whole object. However, I also have to generate distributions with a controllable bias toward a specific class, namely for those sub-objects that do provide information about the class of the object as a whole.

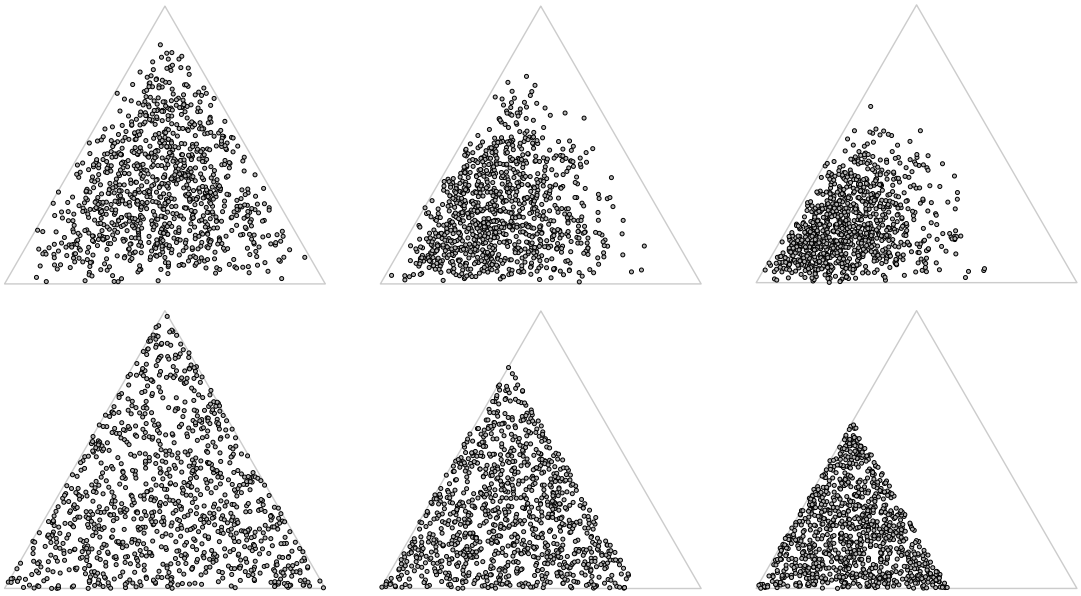


FIGURE 4.4: Each plot shows the sampling of 1000 data points. Top with Dirichlet bias: $Dir(2+b, 2, 2)$ with $b \in [0, 2, 4]$ from left to right. Bottom additive bias: $(1-b) \cdot Dir(1,1,1) + (b,0,0)$ with $b \in [.0, .2, .4]$

A (hyper-)parameter of the Dirichlet distribution can be used to introduce bias towards the real class using a real value b, $b \in \mathbb{R}_+$, with $\vec{\alpha} = (r + b, r, \ldots, r)$. In the following this is referred to as *Dirichlet bias*. Without loss of generality, I will always add the bias to the first class. Alternatively, given a bias b, $b \in [0,1]$, a finite (unbiased) probability distribution can be created first, with the methods outlined above, and then rescaled to sum $(1-b)$. Finally b can be added to one of the classes. This is referred to as *additive bias*.

Figure 4.4 again shows examples with three dimensions (classes). The bias increases from left to right; I chose the bottom left corner to represent the real class. The additive bias (bottom) scales the triangle of sampled points to the allowed area. In this case "allowed" means that the distance to the corner of the real class is at most $1-b$. For $b = 0.4$, which is still smaller than 0.5, the area of allowed points is already smaller than the forbidden area (36% vs. 64%). This visualizes the fact that additive biases that are already small strongly limit the allowed space by the factor $(1-b)^{d-1}$, for d dimensions.

Although the biases cannot be directly compared, this effect seems less strict for the Dirichlet bias as can be seen in the upper plots of Figure 4.4. There is no area that is strictly forbidden, but sampling points far from the real class becomes more and more unlikely.

When b is chosen to be an integer value, the computation is a bit simpler, because then one needs to sample from a Gamma distribution with $\alpha > 1$ and $\beta = 1$. In this case the summation property of the Gamma distribution can be exploited, that is, one sums up α independent samples from an exponential distribution with $\lambda = 1$. Generally, the summation property states that the sum of s independent random variables that are Gamma(α_i, β) distributed, $i = 1, \ldots, s$, is a random variable that is Gamma$(\sum_{i=1}^{s} \alpha_i, \beta)$ distributed.

For integer and non-integer values of α, $\alpha > 1$, and $\beta = 1$, the method of Marsaglia and Tsang [99] may be used.

4.2 Experiments

As described at the beginning of Section 4.1, I tried to keep the assumptions as simple as possible. Hence I assumed that fairly few parameters suffice to model sub-objects:

1. A parameter $v \in [0, 1]$ defines the fraction of sub-objects that carry information about the true class.
2. All sub-objects that do carry information about the true class, carry the same amount of information. This is modeled in form of a fixed bias b toward the true class.
3. In those sub-objects that carry information, all classes other than the true class are treated equally; in those sub-objects that do not carry information about the true class, all classes are treated equally.

The experiments are performed with both the additive bias and the Dirichlet bias which are explained at the end of Section 4.1.

I conducted experiments to answer several questions:

> **Q10:** *How do the different aggregations perform compared to each other? Is there a method that consistently outperforms the others?*
>
> **Q11:** *How do other factors like the number of classes, the underlying distribution, the number of sub-objects, the fraction of information-carrying sub-objects and the strength of the bias towards the true class in these sub-objects influence the accuracy?*

In order to answer these questions, I performed experiments for all combinations of the following parameters:

1. The number of classes was set to 3, 4, 5, or 6.
2. The number of sub-objects was set to 5, 10, 20, 50, 100, 200 or 500. (In the diagrams shown in Figures 4.5 to 4.10 this number is stated in the second title row.)
3. For additive bias, r was set to 0.25, 0.5, 1 or 2 (for $r > 1$ the density was higher for points in the center of the $k-1$ simplex and lower near its corners, for $r < 1$ inverted) and b was set to 0.00, 0.05, 0.10, ..., 0.50, that is, I sampled from $(1-b) \cdot \mathrm{Dir}(r, r, \ldots, r) + (b, 0, \ldots, 0)$.
4. For Dirichlet bias, r was again to 0.25, 0.5, 1 or 2 and b was set to $0, 1, \ldots, 5$, that is, I sampled from $\mathrm{Dir}(r + b, r, \ldots, r)$.
5. All six aggregation methods from Section 2.5 as well as a few combinations of these basic methods were computed. (In the diagrams shown

FIGURE 4.5: Result bar charts: no aggregation for the case of 3 classes and additive bias (left, $((1-b)\cdot \mathrm{Dir}(1,1,1)+(b,0,0))$ or Dirichlet bias (right, $\mathrm{Dir}(2+b,2,2)$).

in Figures 4.5 to 4.10, the aggregation method is shown in the first title row.)

For each of these sets of parameters, 100,000 runs were performed in order to minimize noise effects caused by small numbers of repetitions. Due to their huge number, I only present and discuss the most interesting results in the following. Diagrams for all results are available online[1].

Results Without Aggregation

Figure 4.5 shows the plain version without aggregation: Additive bias on the left and Dirichlet bias on the right. When the bias or the fraction of information carrying sub-objects is zero, the results show an accuracy of 33%, as expected for random guessing with three classes. In preliminary experiments with only 10,000 runs minimal variations could still be observed here. Hence the number of runs was increased to 100,000, resulting in no discernible variation in these values.

For an additive bias of 0.5 and a fraction of information-carrying sub-objects of 100% it is effectively impossible that a class other than the true class is predicted for any sub-object. The reason is that the bias ensures that the probability of the true class is at least 0.5, while the probability of any other class can be 0.5 at most. Theoretically it is possible that the true class as well as one of the other two classes could be both assigned exactly 0.5, while the third class is assigned a probability of 0. In such a case, random selection from the classes with highest probability could choose the wrong class. However, the probability of this happening is effectively 0. As a consequence the accuracy reaches 100% for $v = 100\%$ and $b = 0$, even without any aggregation of subclassifications.

For a Dirichlet bias, correct classification of all sub-objects cannot be guaranteed for $b = 5$ or even higher values (tested in preliminary experiments), as this type of bias only introduces a higher density for higher probabilities, but still allows the probability of the true class to be arbitrarily small. As a

[1] `https://bit.ly/39NyVYD`

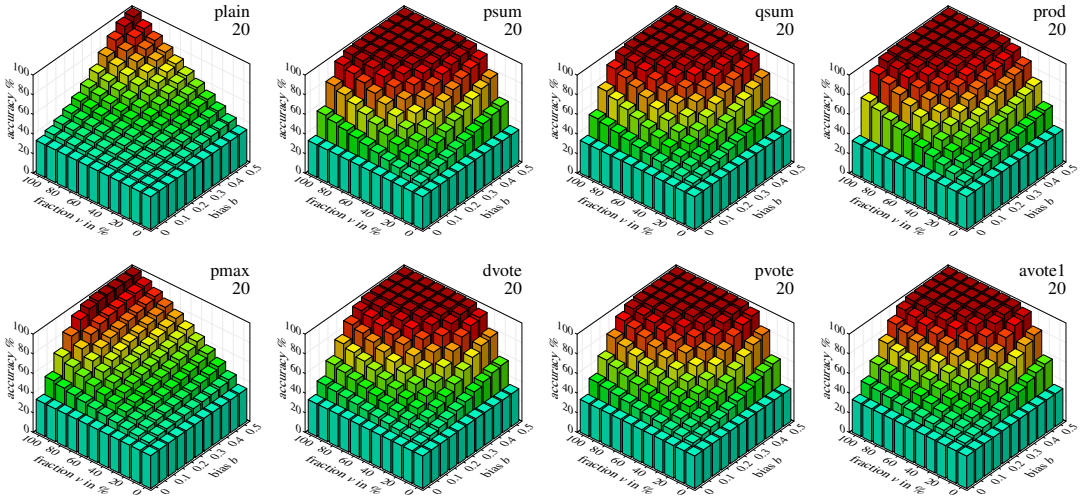


FIGURE 4.6: Different aggregation methods for the case of 3 classes, a distribution of $(1-b)\cdot \mathrm{Dir}(1,1,1)+(b,0,0)$, and 20 sub-objects.

consequence, the accuracy even for the most extreme situation reaches only about 95% here.

Further, both additive bias and Dirichlet bias show the intuitively expected linear growth w.r.t. the fraction of information-carrying sub-objects, best visible for a bias of $b = 0.5$, but also clear for other bias values. With regard to increasing bias (for a fixed fraction), however, the accuracy growth is not linear. For additive bias the growth accelerates first with increasing bias before starting to saturate around $b = 0.35$ or $b = 0.4$. This is plausible, given that the random add-on to the bias for the true class shrinks as less and less probability mass is distributed onto the classes. For Dirichlet bias the accuracy growth generally decelerates for increasing bias, which is plausible given the summation property of the Gamma distribution that underlies the data generation (see Section 4.1).

Result Overview on Aggregations

Figure 4.6 compares the aggregation methods (indicated in the first row of the diagram titles) for 3 classes and 20 sub-objects (indicated in the second row of the diagram titles) in the case of additive bias. For easier comparisons, the diagram for plain (no aggregation) is repeated. Clearly, all actual aggregations perform better, most considerably better. Hence I can conclude that even a bad aggregation is better than no aggregation at all. The worst method is clearly pmax, which is obviously negatively affected by there being sub-objects that do not carry information about the true class. In these cases, due to the data being generated by sampling uniformly from a 3-simplex, the chance of a class other than the true class receiving a (very) high probability is significant. As a single such sub-object can lead to a misclassification, the accuracy can be fairly low. Even for an (additive) bias of 0.5, pmax performs only marginally better than no aggregation at all.

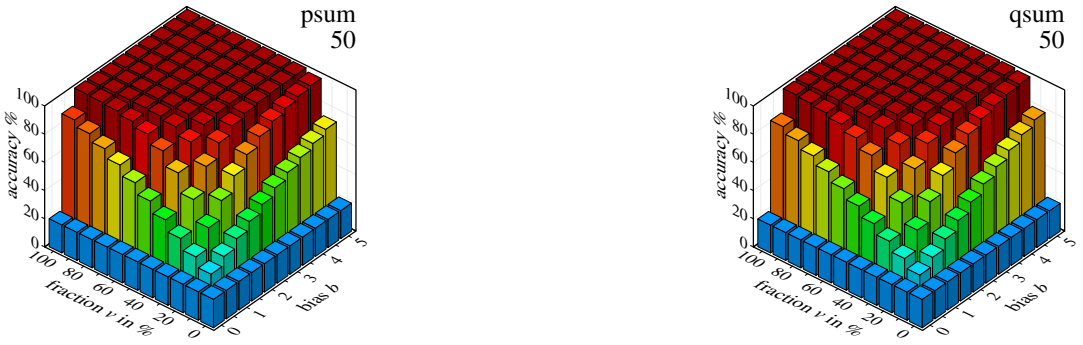


FIGURE 4.7: Sum of probabilities (left) and squared probabilities (right) for 50 sub-objects, 5 classes and a distribution of $\mathrm{Dir}(2+b, 2, 2)$.

Voting Methods

For a fraction of 100%, however, pmax performs slightly better than dvote (majority vote) and avote (majority vote with abstention). All other methods show much better results for high biases, that is, they are much less negatively affected by sub-objects without information about the true class. Probabilistic voting (pvote) performs slightly better than discrete voting (dvote). This is plausible, as the value of the bias is taken directly into account in the probabilistic voting scheme. Voting with abstention differs only minimally from discrete voting.

Product versus Sum

An interesting case is the product (prod): It outperforms all other methods for small biases, which seem to have a fairly strong influence in this aggregation. This effect is clearly stronger for multiplication than for addition, which can be seen by comparing it to psum. Unfortunately, the product is less tolerant to random noise, which can be seen for small fractions of information-carrying sub-objects. As it seems, the variance in the class probabilities for these sub-objects causes more incorrect classifications for a product aggregation, while psum appears to be more robust. This is plausible: What makes prod more sensitive to small biases is exactly that which makes it also more susceptible to random variation.

Sum of Squared Probabilities

Squaring the probabilities before summing up (qsum) consistently performs slightly worse than the non-squared probabilities for small biases. This effect is stronger for more sub-objects and Dirichlet bias, as shown in Figure 4.7. This is also plausible: Squaring the probabilities increases not only the effect of a bias, but also the effects of random noise.

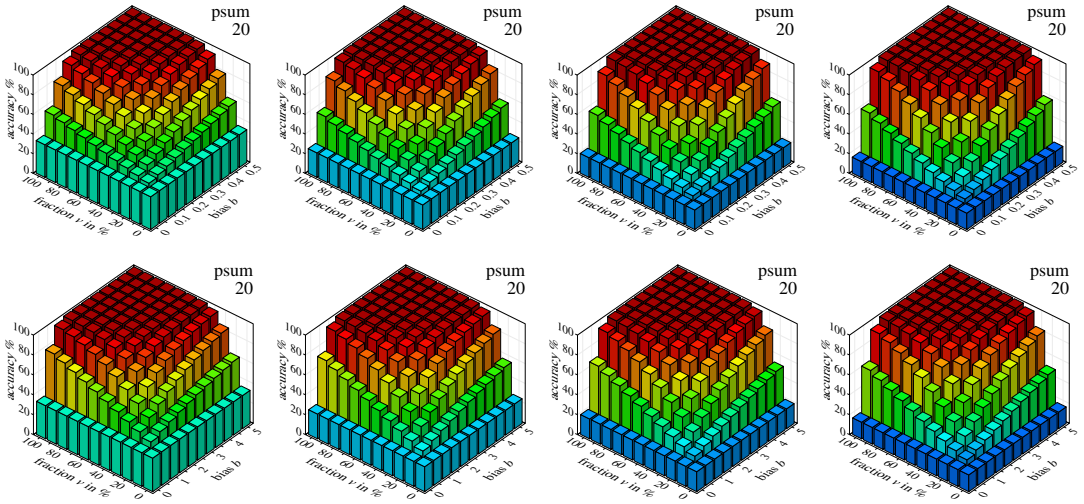


FIGURE 4.8: Behavior of the aggregation method sum of probabilities for 20 sub-objects, 3 to 6 classes (left to right) and two distributions: $(1-b)\cdot \mathrm{Dir}(1,1,\ldots,1) + (b,0,\ldots,0)$ (top row) and $\mathrm{Dir}(2+b,2,\ldots,2)$ (bottom row).

Probabilistic Sum

Figure 4.8 shows the differences between the two forms of bias and for higher numbers of classes for the method psum, which, judging from Figure 4.6, is one of the best methods. From left to right the number of classes is increased, which can be easily seen by the reduced accuracy values for random guessing (i.e., for $b = 0$ or $v = 0$). The top row shows additive bias, where for more classes higher accuracy values are achieved. A fixed bias value has a stronger effect when there are more classes because more classes share the remaining probabilities, so the relative bias increases.

An opposite effect occurs for the Dirichlet bias in the bottom row: Here, the Gamma distribution reaches a higher value, but the divisor also has a high value too, as it is computed as the sum over all classes. Therefore, the relative bias decreases for more classes.

Maximal Probability

Figure 4.9 again compares the effects of the number of classes and both forms of bias, but here for the worst method pmax. The higher the number of classes, the less relevant the type of bias, as the diagrams become more and more similar. For an increasing number of classes, pmax becomes worse for small biases, as with the increased number of remaining classes more possibilities exist that may have probabilities beating the true class whose bias is small. For a strong additive bias and an increasing number of classes, the accuracy improves. Again, this is caused by the effect that the remaining classes share the remaining probability mass and thus the chances that any of them can beat the true class decrease. For the Dirichlet bias, there exists no such effect as the class probability distribution tends toward a uniform distri-

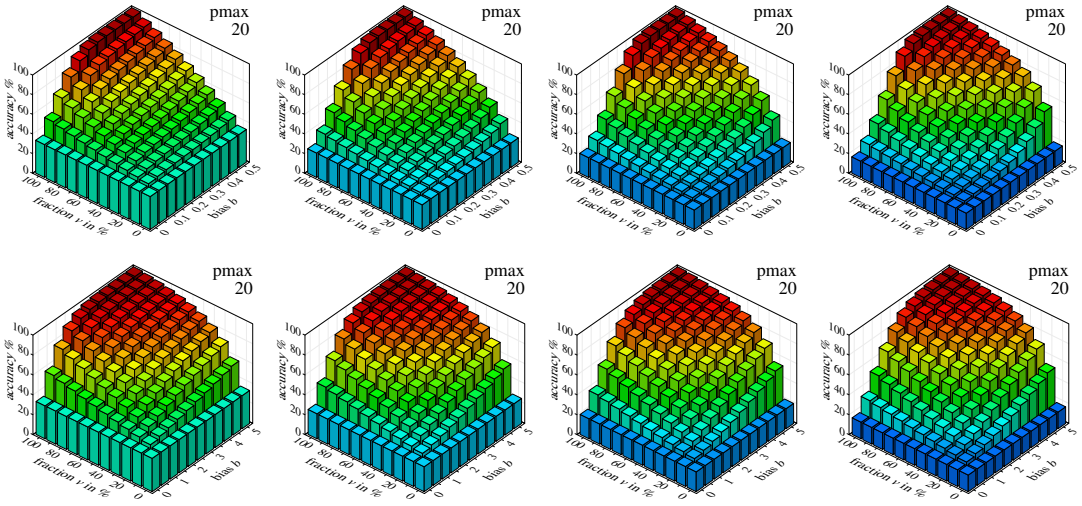


FIGURE 4.9: Behavior of the aggregation method maximum of probabilities for 20 sub-objects, 3 to 6 classes (left to right) and two distributions:
top row: $(1-b)\cdot \mathrm{Dir}(1,1,\ldots,1)+(b,0,\ldots,0)$, bottom row: $\mathrm{Dir}(2+b,2,\ldots,2)$.

bution. Thus, with a Dirichlet bias pmax reaches generally higher accuracies than with additive bias, especially for only few classes.

Combined Aggregation Functions

So far it would appear that there is not always a clear order for the different aggregation functions. For some parameters one is better, while for others it is worse. Finally I tried combinations of the basic aggregation methods that performed best in some cases hoping to create a new aggregation that shared the best properties of both base functions. Those generally well performing basic methods are clearly psum and prod.

> **Q12:** *Can (linear) combinations of well performing aggregation functions preserve the best of both functions?*

I consider weighted sums of these two methods now. Figure 4.10 shows results generated for 50 sub-objects and a Dirichlet bias. The left row contains the two basic aggregations psum and prod, with prod performing slightly better for a small bias and psum slightly better for small fractions. The second row shows the results for an aggregation that combines prod and psum as follows:

$$c = \arg\max_{j=1}^{k}(\textstyle\sum_{i=1} \ln p_{ij} + \beta \sum_{i=1} p_{ij})$$

for $\beta = 5$ (ppmix1) and $\beta = 10$ (ppmix2). See also Section 2.5.

In the following, I show several combinations of the basic aggregation methods with a simple voting (or averaging) scheme. These are shown in the right column of Figure 4.10, namely a combination of psum, prod, pmax, pvote and avote2 ($a = 0.2$), labeled stack1, and a combination of psum, prod and avote2. Unfortunately all of these combinations merely average the accuracies of the basic methods and are unable to combine their strengths. That is,

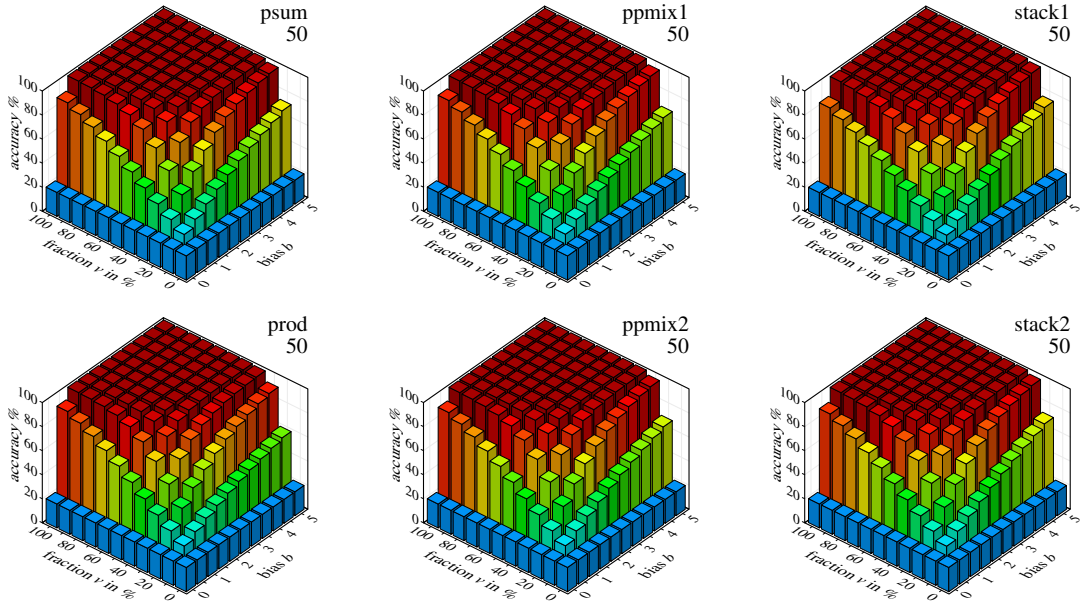


FIGURE 4.10: Behavior of the aggregation method maximum of probabilities for 50 sub-objects, 5 classes, distribution of $\mathrm{Dir}(2+b, 2, 2, 2, 2)$ and 4 different combinations of basic aggregation methods.

the weak spots of the basic methods are improved, but at the price of deteriorating their strong parts. This is plausible, though: If a method is more sensitive to small biases, it is also more sensitive to the randomness in the distributions for the information-less sub-objects and thus suffers from a reduced accuracy for small fractions.

Underlying Distributions

In the figures above, Dirichlet bias means $\mathrm{Dir}(r+b, r, \ldots, r)$ for $r = 2$, and additive bias uses $r = 1$ in the equation $(1-b) \cdot \mathrm{Dir}(r, \ldots, r) + (b, 0, 0 \ldots)$. In the following, I vary the parameter r for both distributions. Figure 4.11 shows variations of the additive bias, when r is varied (top: $r = 0.25$, middle: $r = 1.0$, bottom: $r = 2.0$). The baseline without aggregation (plain) is shown on the left. For $r = 0.25$ a high bias is needed, in order to gain good accuracies. Small biases bring results close to random guessing, even if the fraction v is high. For increased r this behavior is reduced. But also using the product as aggregation method is very efficient. Although for example for $v = 100\%$ and $b = 0.05$ the plain prediction is only slightly better than random guessing, the product is able to achieve 100% accuracy. Psum reaches nearly 60%. Similar as in the cases seen before, psum is the better choice for high biases and small fractions.

Figure 4.12 shows the results for the Dirichlet bias $\mathrm{Dir}(r+b, r, \ldots, r)$ when r is varied (again top: $r = 0.25$, middle: $r = 1.0$, bottom: $r = 2.0$). Visually, there is only a small difference for varying r. In contrast to the additive bias, here, a smaller r seems to generally increase the performance for plain, psum and prod. This effect is best visible for small biases and high fractions. For high

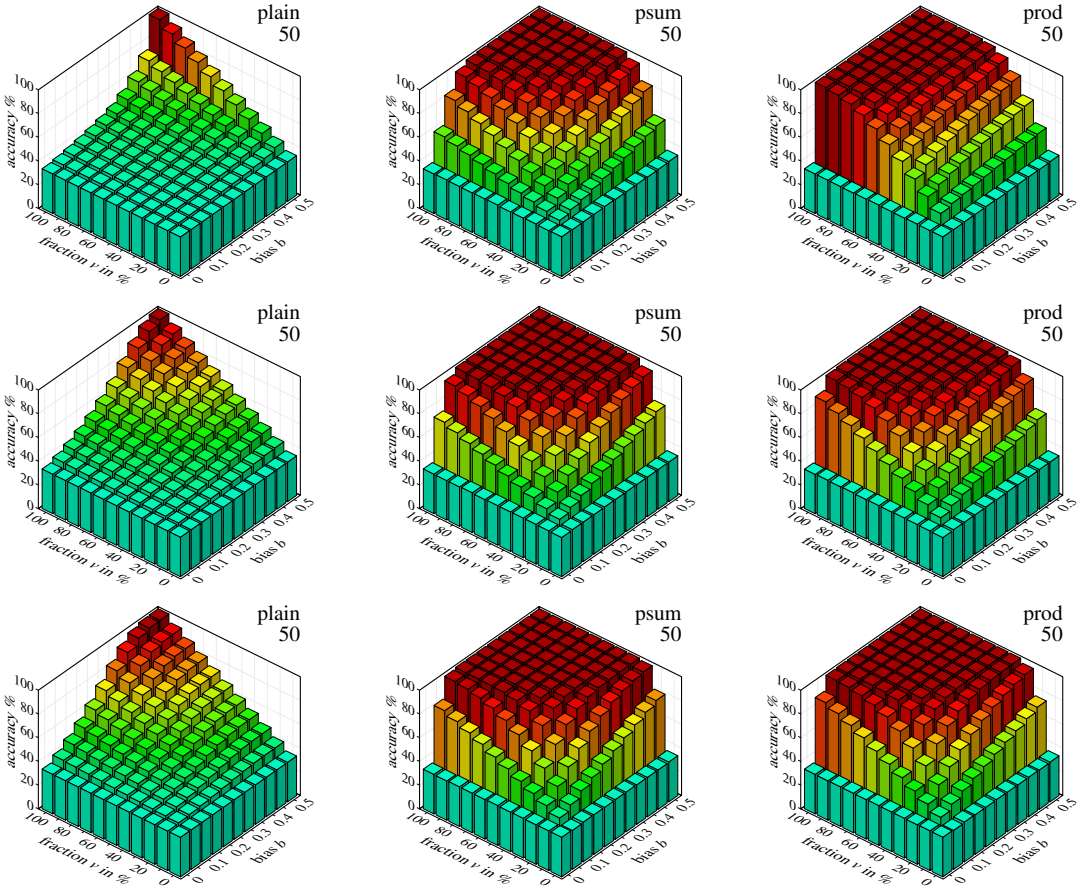


FIGURE 4.11: Behavior of plain, psum and prod for additive bias $(1-b)\cdot \mathrm{Dir}(r,\ldots,r)+(b,0,0\ldots)$ when parameter r is varied $(0.25, 1.0, 2.0$, top to bottom)

biases, the value of r seems to have only a minor influence on the accuracy. Because of the good overall performance, there are also only small differences between the aggregation functions. The overall trend can be verified once again: psum is better for small fractions, and prod is better for small biases.

So far, I have compared pairs of plots and explained for which parameters some methods are better than others. Although this has given valuable insights, such that psum and prod mostly perform better than other methods, it is difficult to gain an overview this way.

Domination Graphs

Comparing pairs of aggregation functions by their bar charts is a tedious task. However, it is helpful to visualize which methods outperform others, and when this is the case. If one function is always better than another one, this would however be an helpful information. To obtain this information — it is ironical — I need to *aggregate* my *results*. Figure 4.13 shows a graph with such a summary. Nodes represent the aggregation functions.

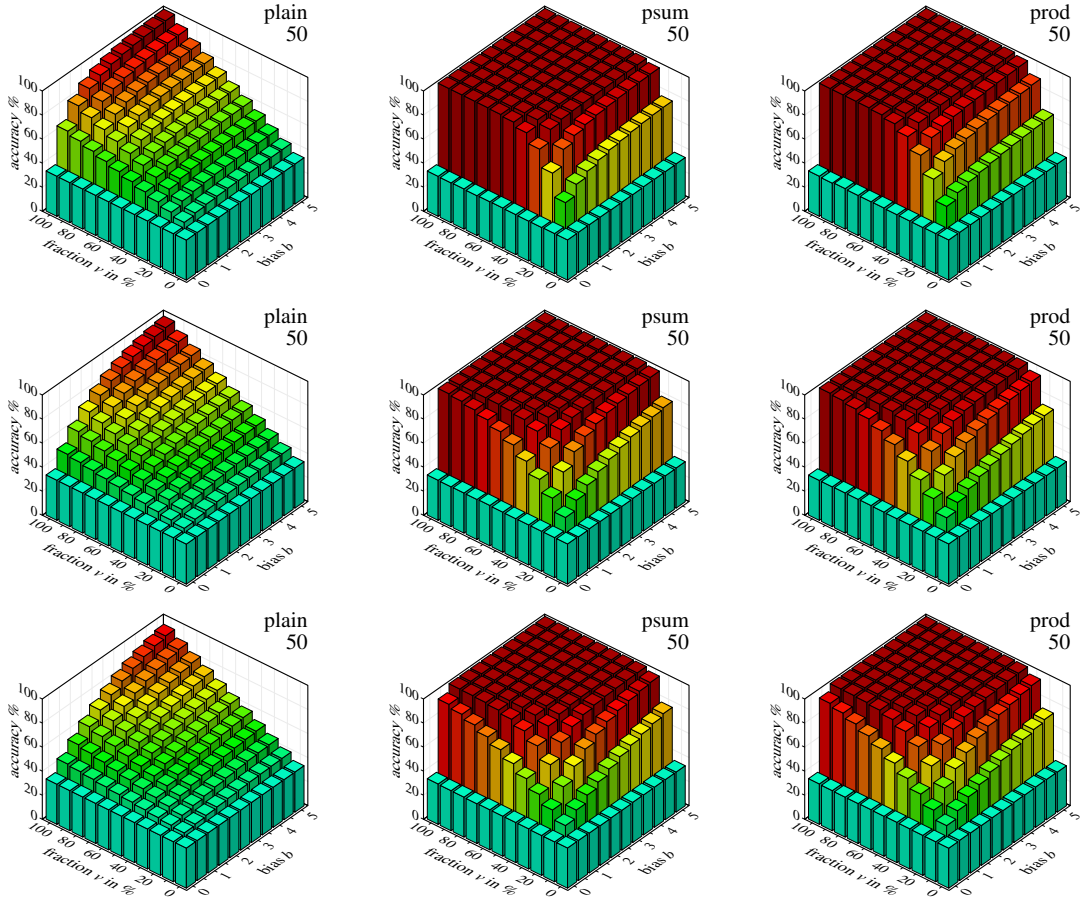


FIGURE 4.12: Behavior of plain, psum and prod for Dirichlet bias $\mathrm{Dir}(r+b, r, \ldots, r)$ when parameter r is varied (0.25, 1.0, 2.0, top to bottom)

A directed edge points from f_1 to f_2 iff *for all* combinations of fraction v and bias b $acc(f_1) \leq acc(f_2)$, in other words f_1 is dominated by f_2. The domination graph gives no information which set of parameters caused the absence of an edge. Here, I limit the graph to the seven basic functions and omit the node for no aggregation (plain) as it would add edges to all (other) functions. This would show again, that any aggregation is better than no aggregation. Such a graph represents the results when several parameters are fixed and which are given in the heading: the value of r (here 2.0), the type of bias (here $p = 2$: Dirichlet, otherwise $p = 1$: additive bias), the number of classes (here 3), and the number of snippets (500).

The measured accuracies show only an approximation for $n = 100,000$ runs and thus underlie natural fluctuations as n is finite. In the bar charts (Figures 4.5 to 4.10), these fluctuations exist as well, but are not visible because they are too small. However, because of fluctuations comparing the values without tolerance would not yield any dominated solutions. Thus I compare if $acc(f_1) \leq acc(f_2) + \varepsilon$ for all combinations of bias b and ratio v.

How to choose ε? For $b = 0$ or $v = 0$, the result should resemble random guessing; for $n \to \infty$ $acc(f_j) = \frac{1}{c}$ for all aggregation functions f_j. From all these results of chance, I choose the difference of the smallest and biggest

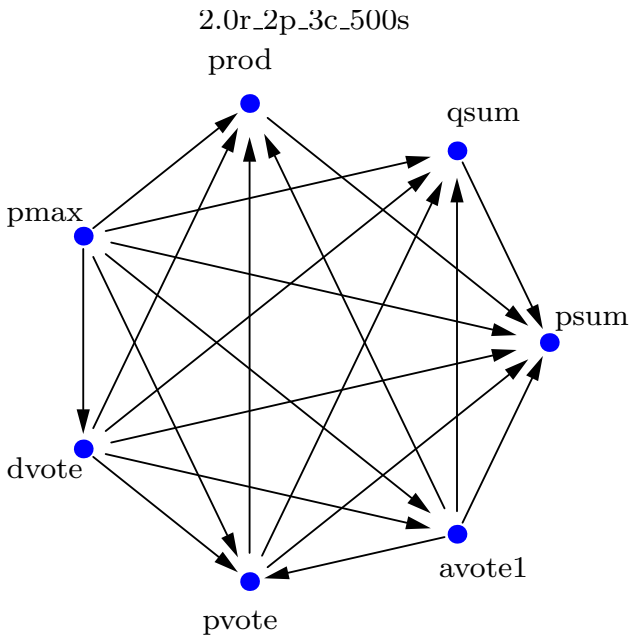


FIGURE 4.13: Behavior of the basic aggregation methods for 500 sub-objects, 3 classes and Dirichlet bias

value as ε. A tolerance of 0.01113 suffices to guarantee that random fluctuations do not interfere. Just to be completely sure, I added another 20% of additional tolerance and set $\varepsilon = 1.2 \cdot 0.01113 = 0.013356$.

Because of the tolerance it is possible that edges exist in both directions $f_i \mapsto f_j$ and $f_j \mapsto f_i$ if both functions perform similarly (with regard to ε). Fortunately, this rarely happens in our data. (One example is depicted in the upper right graph of Figure 4.15 between avote1 and dvote.)

Figure 4.14 gives an overview of how the graph changes for 20, 100 or 500 snippets from left to right and how it changes for more classes (3 to 6) from top to bottom, for Dirichlet bias with $r = 2$. The graph with the least edges is the one for six classes and 20 snippets (bottom left). Although each graph shows results from independent experiments — all edges of one graph also exist in the graphs for more snippets or fewer classes. This is a plausible effect: With fewer classes, the chances of choosing the wrong class by chance decrease. Further, a higher number of snippets also increases the absolute number of snippets that contain information about the real class.

Another effect can be visually verified in Figure 4.14: With few snippets or many classes, psum and prod are the only nodes that have no outgoing edges, so there are parameters b and v, such that they are the optimal choice. Additionally, for 500 snippets or 100 snippets and 3 classes, prod is also dominated by psum. This effect occurs only for Dirichlet bias and shows up as well for fewer snippets when r is smaller.

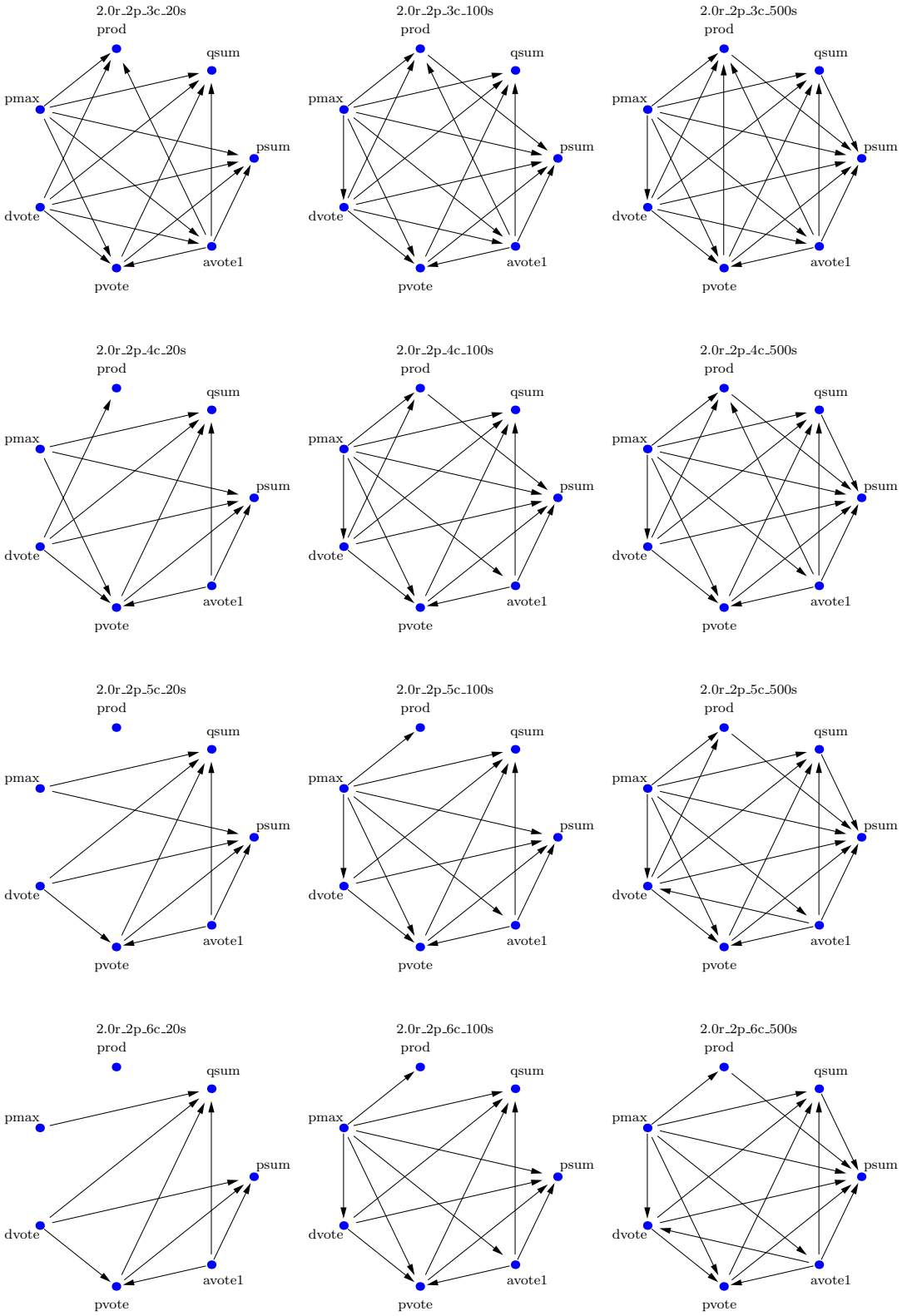


FIGURE 4.14: Behavior of the basic aggregation methods for 20, 100 and 500 sub-objects (left to right), 3, 4, 5, 6 classes (top to bottom), Dirichlet bias, $r = 2.0$

For $r = 0.5$ this is shown in the appendix in Figure A.1 (p. 178). In these cases, psum is the overall best aggregation function. Figures showing results for various values of r are available online[2].

I recommend avoiding pmax because it has no incoming edges at all. Interestingly, pvote dominates avote1 and dvote in all of the cases.

Figure 4.15 shows the overview when the additive bias is used. One of the two effects of Dirichlet bias are also visible there: With more snippets, more functions are dominated. But the effect for the number of classes is reversed: More classes often cause more edges. Note that the results here are not always consistent. For example, sometimes dvote dominates pmax, sometimes it does not. However, the main findings still hold: pmax is always dominated by at least one function; prod and psum perform excellently. In contrast to Dirichlet bias, prod is never dominated by psum, not even for 500 snippets. This holds also for smaller values of r (not shown here). Again, pvote dominates avote1 and dvote in all of the figures. The additional information contained in the probability of pvote seems to add helpful information to the overall decision.

The comparison between avote1 and dvote is interesting: Independent of the number of classes, whenever the number of snippets is high (here 500), dvote dominates avote1. In dvote, the number of votes for the wrong classes cancel each other out and suffice to generate a better overall result than avote1. Abstention holds back the small pieces of information that would be needed for the correct prediction. When there are many votes close to equal probabilities ($p \approx \frac{1}{c}, \ldots, \frac{1}{c}$), but there is a clear trend within these votes, this trend improves the accuracy. On the other hand for three classes, avote1 dominates dvote. So for 500 snippets and 3 classes they dominate each other and thus perform similarly.

Note that due to the *Poincaré Paradoxon* [117] the graph may lose its transitivity property for any tolerance level $\varepsilon > 0$. So the non-intuitive case can occur that there are edges $f_1 \mapsto f_2 \mapsto f_3$ but no edge $f_1 \mapsto f_3$. Fortunately, none of the experiments with the basic aggregation function shows such a case. This reassures that the chosen ε is not too big.

I also generated the domination graphs for the combined aggregation functions. For additive bias, many classes and few snippets no function is dominated by another and the other functions show no clear tendencies (so I omit the figures here). Dirichlet bias is more interesting here and thus shown in Figure A.2 in Appendix A. Functions perform so similar that they frequently dominate each other. An earlier result for Dirichlet bias still (mostly) holds: For more snippets and fewer classes the edges are preserved, also for the combined functions. One exception is the edge from stack2 to ppmix for four classes, which is missing for 100 snippets. I assume this is caused by chance.

[2] `https://bit.ly/39NyVYD`

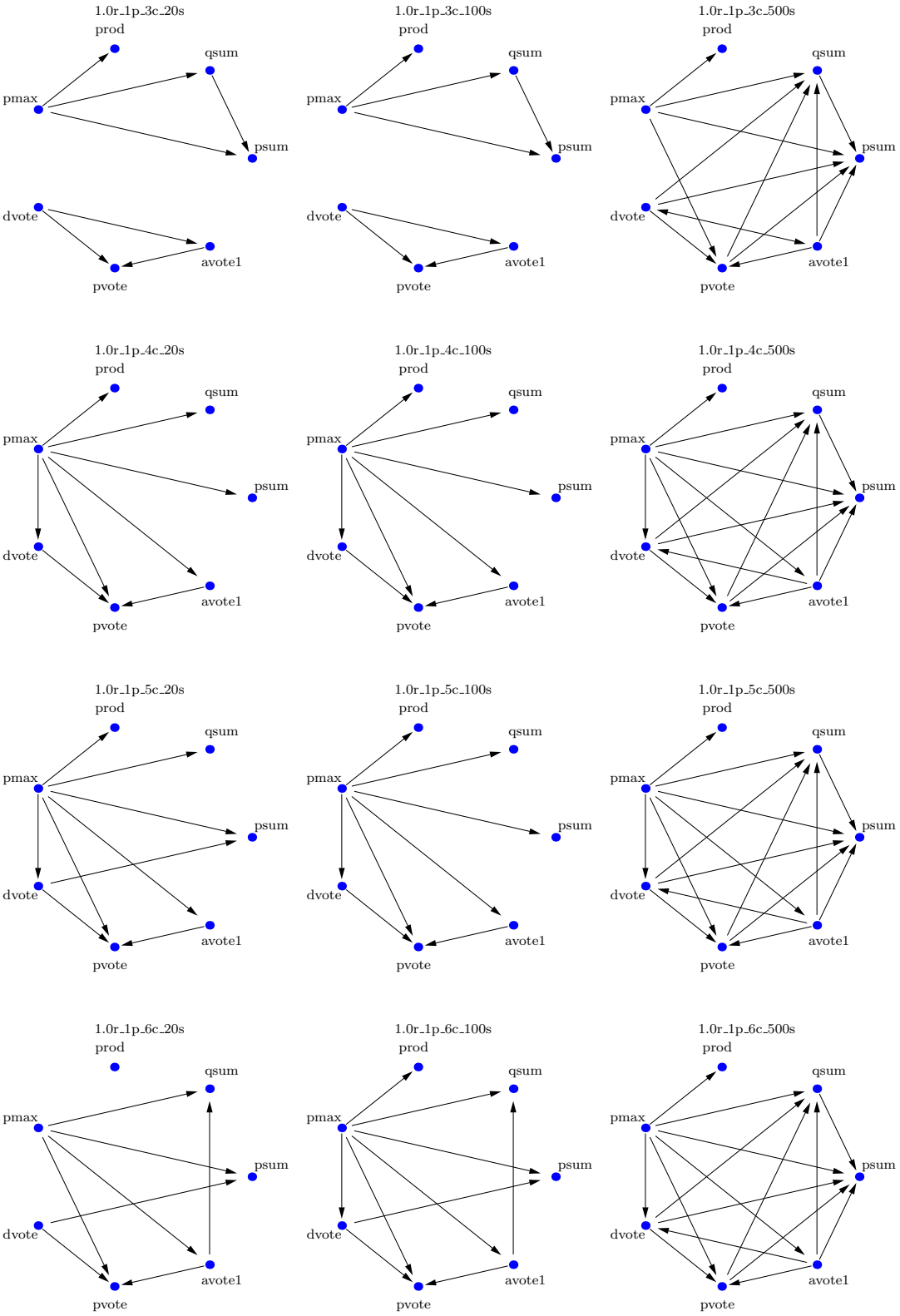


FIGURE 4.15: Behavior of the basic aggregation methods for 20, 100 and 500 sub-objects (left to right), 3, 4, 5, 6 classes (top to bottom), additive bias, $r = 1.0$

Stack2 and psum show an interesting behavior. For 5 or 6 classes stack2 dominates psum; for fewer classes and for 5 classes and 500 snippets they dominate each other. ppmix2 dominates ppmix1 in all cases, so higher weighting on *psum* seems beneficial. In summary psum remains the best method in most cases. Here, ppmix2 outperforms prod — ppmix2 is the only combined function which is able to sometimes keep the best of both basic aggregations[3].

The Poincaré paradoxon can be observed multiple times, in Figure A.2 for example for four classes and 20 snippets:
$prod \leq ppmix2$ and $ppmix2 \leq psum$, but $prod \not\leq psum$.

4.3 Summary

Aggregation functions are not only of theoretical interest, but can also help to boost classification accuracy in practice, by combining subclassifications.

> **Answer** (**Q10**, p. 88): For simulated data, there is no clear best *general* aggregation function, as it depends on the underlying — and in practice often unknown — distribution of sub-objects.
>
> **Answer**(**Q10,Q11**, p. 88): Methods *psum* and *prod* perform best for the tested additive and Dirichlet biases, and are the functions to recommend: *psum* for small fractions or for many snippets and few classes and *prod* for small biases. When proper values or estimates for type of bias and its parameters are given, then the aggregation function should be chosen accordingly.

I also compared voting schemes for multiple classes: They usually performed worse and might be better suited for other distributions of sub-objects. A core result of my investigation is that it appears to be impossible to have good performance for small biases and at the same time for small fractions, since any method that is sensitive to small bias is also sensitive to the random noise for information-less sub-objects.

> **Answer** (**Q12**, p. 93): Most linear combinations of *prod* and *psum* were not able to preserve the best properties of both functions. For Dirichlet bias *psum* works extraordinarily well and can even dominate *prod*. Hence, the combination of prod and *psum* with high weight on *psum* (*ppmix2*) is also able to dominate *prod* in this case.

The source code for the experiments is publicly available as Python script `clsagg.py`[4]. For experiments with non-integer values of r, use `clsagg_np.py`. Explanations of the implementation are given in Section 4 of our paper [34].

[3] I implemented an additional level of aggregation, where I count the numbers of edges of several domination graphs to see which functions frequently outperform others. The sums are presented as heatmaps and are available online `https://bit.ly/39NyVYD`.

[4] Available together with scripts to parallelize the execution of multiple experiments and to generate result bar charts (as they are shown in Section 4.2) in the archive at `https://www.borgelt.net/python/clsagg.zip`

The code is easily extendable to enable researchers to implement their own aggregation functions and data distributions (especially biases), as well as conduct further experiments (e.g., for more classes).

Chapter 5

EEG Smoker Data Set

I investigate the differences in brain signals of craving smokers, non-craving smokers, and non-smokers. To this end, I use data from resting-state EEG measurements to train predictive models to distinguish these three groups. I already motivated in Chapter 1 that this task is fundamental research in the field of addiction.

Parts of the following chapter are already published in two of my own works [38, 35]. I also describe parts of the Master's thesis of Cedrik Pätz [112] that I supervised. He was the first who worked on this data set and I use some of his results when I create my own models.

I first explain how the data was recorded, then present the preprocessing, which was performed in cooperation with Cedrik Pätz and Sarah Donohue. Further, I describe particular difficulties of this data set, and with which methods I handled them. First, I use simple models: majority class prediction, random guessing, linear discriminant analysis and naïve Bayes. Second, I apply various neural network models to answer the questions about the optimal choice of the model structure, sampling techniques and aggregation methods. Later I use the results to answer questions about the data. A full list of questions is given at the beginning of Section 5.4.

In Section 5.5, I describe my implementation of a tool that visualizes the input data as well as the resulting predictions. This tool is meant to enable experts to understand the trained models and thereby gain insights into which features differ between addicted and non-addicted people. Finally, I present results on permutation tests, meant to verify that the model accuracy differs from what is achievable for randomized class labels.

5.1 Data Description

The data stems from a study conducted between 2013 and 2014 at the Leibniz Institute for Neurobiology in conjunction with the Clinic for Neurology at the University of Magdeburg. 30 smokers participated and stated that they were long term smokers, smoking at least ten cigarettes per day for at least the past half year. 9 further participants were non-smokers. The study for which

the data where initially collected [39] used EEG data that were time-locked to the onset of smoking-related and non-smoking-related images. After these experiments the participants came to a rest and the timer for the resting state signal was started afterwards, taking signals for around 13 minutes. Resting state means here that the subjects have their eyes open and no task to perform. They were instructed to fixate a cross to limit their eye movements and were asked to move as little as possible during the measurement. Each of the 9 non-smoker was measured once, while all 30 smoker were measured twice: once craving, after an abstinence of at least 4 hours and once non-craving, having smoked directly before the measurement started. A full list of measurements is given in `data/SmokingRestingStateSubjects.csv`. It contains the subject id, the craving state, the age and the handedness, as well as the score of a Fragerström Test for Nicotine Dependence (FTND) [61]. As this test asks about smoking habits, non-smokers automatically receive a score of 0.

The Ethics Committee at the Otto-von-Guericke University of Magdeburg authorized the experimental methods and procedures used in this study. All participants of the study gave their written, informed consent prior the participation. All subjects were financially compensated for their time.

Due to the small number of participants, the general reliability of the results generated from this data set is low. The results should be verified with samples from more subjects, when available.

Effects Hypothesized to be Inherent

I suppose that two effects could be inherent, an effect of *addiction* and an effect of *craving*. *Addiction* would be visible if models were able to distinguish between smokers and non-smokers. The effect of *craving* would show up, when comparing craving smokers and non-craving smokers.

> **Q13:** *Can models distinguish between smokers and non-smokers?*
>
> **Q14:** *Can models distinguish between craving smokers and non-craving smokers?*

I investigate these effects by creating models for distinguishing each pair of the two classes for all three classes. Considering *craving smokers vs. non-smokers* should measure both effects, which, if these effects sum up, would be indicated by a high predictability. *Non-craving smokers vs. non-smokers* only measures the effect of addiction and *craving vs. non-craving* takes the effect of craving into account. The most difficult problem uses data from all three classes and tries to distinguish them all.

Subject Dependency

The measurements for craving and non-craving were taken from the same subjects. This is a problem with the assumption of test sets and training sets

being independent and identically distributed (i.i.d). The problem occurs when data from the same subject are used in both sets — training and test set — simultaneously. The session does not matter. For a detailed description, be referred to the work of Le Boudec [87]. Although this seems like a theoretical problem, it is possible that models find and learn person-specific patterns (i.e., identifying a specific subject) [127]. These patterns could confuse the model when a subject was in both data sets at the same time. This gives the model the opportunity to learn individual-specific patterns in the EEG signals, which might be used to identify the person. In this case, the model would recognize a person it was trained on e.g., when craving and would consequently predict craving when the non-craving session is shown. As consequence, it would systematically predict the wrong of the two smoker classes. Note that I empirically verify this behavior in my experiments and show the results in Appendix B.2.

To mitigate this problem, for all subjects both measurements (craving and non-craving) were used either for training or for testing. In this case it is still possible that the model learns person-specific patterns, but these will not directly affect the results. When not explicitly mentioned otherwise, my experiments were performed with splits that respect subject dependency, such that this problem does not occur.

Recording the EEG Signal

The EEG data were acquired using 32 electrodes positioned on the scalp with the right mastoid (the bone behind the ear) as reference — see Figure 2.8 for an example of the used layout. The channels are located according to the 10-20-system described earlier in Section 2.1. The sampling frequency for all measurements was identical (508 Hz), thus no resampling is needed (in contrast to e.g., the data in Chapter 6, where data are taken with different sampling frequencies). Here, the signal was lowpass-filtered online at 50 Hz, with low impedances (5 kΩ) maintained for each channel. In order to remove noisy parts at the beginning and the end, the measurements were cut to a fixed length of 9.5 minutes. In total, each measurement contains 508 Hz $\times$ 60 seconds $\times$ 9.5 minutes $\approx 290,000$ measuring points per channel.

5.2 Difficulties with this Data Set

From the data analysis perspective, 36 participants is very few — prominent tasks as described in Section 3.3 make use of *big* data with millions of samples. The features suited for the task at hand are unknown; earlier results that considered differing frequency bands to separate smokers and non-smokers showed inconsistent results. In contrast to these methods that use static features, I first use simple machine learning models to create a base line and later I apply neural networks that can create their own features. Note that

	craving smokers	non-craving smokers	non-smokers
Overall	27	27	9
Training	21	21	7
Testing	6	6	2

TABLE 5.1: Numbers of measurements per split

in the machine learning approach, the EEG signal is split into smaller parts I call snippets, which are used as training data.

Class Imbalance

With 3 times as many smokers as non-smokers, the classes are unbalanced. I handle this by balancing the class weights during the training and the validation process. To score the results, I use the class-balanced accuracy in all experiments, as described in Section 2.3.3.

Validation

I described different validation techniques in Section 2.3.2. Pätz applied a Leave-One-Out Cross-Validation *(LOO-CV)* for this task. So he trained his models on samples from all but one subject, for all 36 possibilities. This method makes best use of the available data by maximizing the number of training samples, but also has some drawbacks: It lacks the possibility to perform more independent runs and covers only a very local area in the model space and thus the reliability of the results is limited. With test data from only one person, huge fluctuations are to be expected.

The question for the optimal split is especially difficult for this data set: 27 smokers and 9 non-smokers are to be split into training validation and test. As there are three times as many smokers as non-smokers, one can focus on a solution for the non-smokers and take the values tripled for smokers without any rounding problems. Using a single non-smoker in a split seems too few because of the overfitting. But with two people in the validation and the test set, only five would remain for training. After long discussions with several experts, I decided to use no validation set at all, two non-smokers and six smokers for testing. The generated split is clarified in Table 5.1.

I generate the splits randomly because of the advantage that it can easily be repeated to generate more results and thus creates more reliable conclusions. In this example, there are $\binom{27}{6}$ possibilities to choose smokers and another $\binom{9}{2}$ for non-smokers, which yields a total of $10,656,360$ possibilities, from which I choose 100 randomly, one for each run. In order to guarantee the same chances for all models, the same 100 splits were used for all of the models.

One drawback of this procedure is that the number of times a patient is in the test set may vary. Thus, when summarizing over several runs, there are

fluctuations per subject — some are used more often for training, others for testing. The numbers of occurrences in the test set for each subject is given in Table 5.2. Because of subject-dependency, smokers are listed only once. Both measurements are used equally often. The average number of occurrences is 22.22 in each class. But some subjects vary much from that value, for example the non-smokers *vb79*, with 30 and *wr15* with only 15 occurrences. As future work I consider an improved algorithm, that returns a set of training (,validation) and test sets, such that the total number of occurrences of each patient in each set is also stratified.

Bootstrapping

Bootstrapping splits each measurement into several snippets of fixed length and samples from them. The model is trained and tested on these snippets instead of the full measurement. This has two advantages: it decreases the (time) dimensionality and increases the number of training samples. Unfortunately, when handling snippets, subject dependency has to be respected, too, since two snippets from the same subject have to be considered dependent. Thus I adapt the sampling process to guarantee independent splits.

There are two open questions regarding snippets: Should snippets be allowed to overlap? How long should the snippets be? A larger window length gives the model a longer signal to process and therefore more information, which might help to distinguish the classes, but it also increases the time needed to process the data. On the other hand a smaller window length makes it possible to generate more training samples.

Overlapping is a natural form of data augmentation, since it enables more training samples. Pätz [112] evaluated different snippet lengths for neural network models on the same data set, without overlapping snippets. He worked on the easier problem with two classes: craving vs. non-smoker and performed a Leave-One-Out Cross-Validation. Figure 5.1 shows training and test accuracy for one exemplary run from the same subject. It shows the graphs for snippets of length 100, 1,000 and 10,000 from top to bottom. An epoch processes all training data (290,000 time steps per measurement) once, independent of the snippet length. The training accuracy increases during training in all cases. The test accuracy for length 100 does not surpass 54%, indicating that the model is unable to generalize properly.

For a length of 1,000 the training works properly and reaches up to 65% class-balanced accuracy. For length 10,000 the test accuracy shows strong fluctuations and a trend to decrease during training, probably because of overfitting. I conclude to use snippets of length of 1,024 for my own experiments. This corresponds to approximately two seconds and can easily be processed with pooling operations without generating rounding errors. Without overlapping this creates $290.000/1.024 \approx 283$ training samples per measurement. With a total of 63 measurements this yields $17,829$ samples, each with 25,600 values, which I use in most of my experiments. Note that in later experi-

person id	occurrences in test set
smokers	
ab82	23
ac06	20
ac49	19
az54	26
bi11	24
dx15	26
eb80	20
fb03	24
ic49	21
js94	20
jy74	14
kj04	20
me68	27
nw43	20
nx50	22
pu11	19
qe30	20
qq31	17
rx94	22
so70	28
ta61	28
tt42	30
wl19	18
wn86	20
wt71	20
ye65	27
zf01	25
non-smokers	
bd53	19
bh27	27
bx95	29
dn20	17
ez57	20
ox81	23
sc17	20
vb79	30
wr50	15

TABLE 5.2: People's number of occurrences within the 100 splits of test set

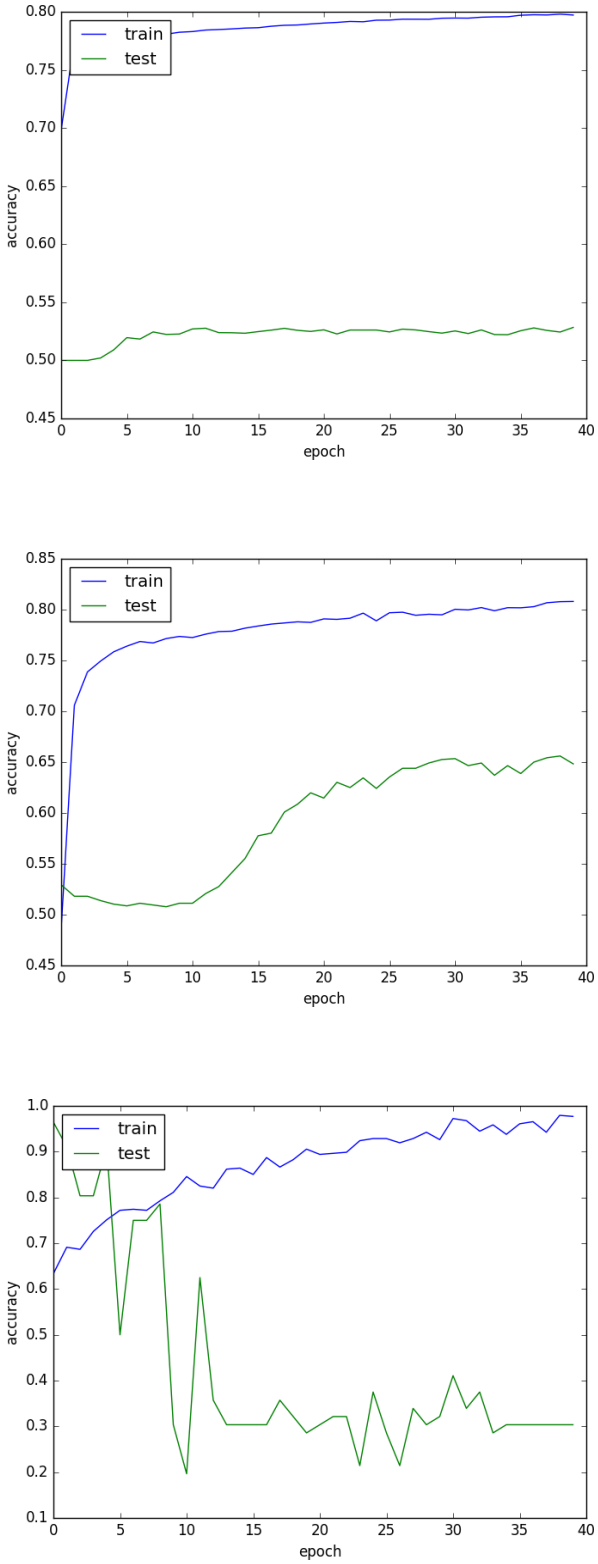


FIGURE 5.1: Comparison of snippet lengths: 100, 1,000, 10,000 from top to bottom (Source: [112, p.50])

ments (described in Section 5.5), I apply a variation on the test set that I call *extensive testing*. It uses *all* possible snippets and allows overlaps.

Sampling

When the split and window size are chosen, I finally need to specify from which position of the measurement snippets are sampled for the training process. I experimented with two possibilities: a sliding window approach and random sampling. The sliding window works deterministically. Only one parameter needs to be set to define the strength of overlapping. This determinism has the advantage that pre-computations like a Fourier transformation or feature generation can be performed prior to training.

If the size of the created features and their number are sufficiently small, then the result of the pre-computation can be saved and thus needs to be performed only once. It can later be used again for training runs of other models or varying training parameters.

Random sampling takes a snippet randomly with replacement from an arbitrary position. Without limitations for the positions, this allows very many different snippets. Saving the result of all of their pre-computations is hardly possible and thus needs to be performed for each training run, which takes additional time. Note that although the starting points of the snippets are uniformly sampled, because of the window length parts near the start and the end have a smaller chance of being presented to the model. This processing maximizes the number of possible training snippets. And more snippets during training hopefully increases the model's ability to generalize. Pätz also describes experiments that apply a Fourier transformation as pre-computation, which seems to be one reason for him to choose sliding window as sampling method.

> **Q15:** *Which sampling technique performs better? A sliding window or a random sampling?*

Variance Minimization

Generally, there is an interest in models with high mean prediction values with small variance. For two models with identical mean value normally, the one with smaller variance is preferred, because its predictions are more consistent. Given the sample size, here I aim for a methodology that adds as little variance as possible.

In my experiments, there are two sources for variance: First, variance caused by different splits $Var(S)$ and second, variance caused by different weight initializations $Var(W)$. Those two cannot be distinguished when considering only results from experiments influenced by both factors as these experiments measure $Var(W + S)$. But the second one can be isolated by performing multiple runs on a fixed split. Assuming independence of these effects,

the variance caused by the weights can be estimated [43, p.233]:

$$Var(W) = Var(W + S) - Var(S)$$

It is also possible to avoid the effect of variance in the weight initialization by performing the initialization not once per run, but only once in total. However, the Keras framework [23] performs the initialization on the GPU in a way that the values are not 100% deterministic due to race conditions. The only way I found was to initialize the network once, save it and load it at the beginning of each run. However, as I can set $Var(S) = 0$ by using the same split several times, I can estimate $Var(W)$ instead of measuring it.

Implementation of Grouped Batch-Stratification

Given is a set of samples, each consisting of independent and one dependent variable. Then a split function is used to divide this set into subsets, such that the distributions of the dependent variable of the subsets and the original set are identical. In a folded validation, the number of subsets is the number of folds. These folds are then used to create training and test sets for the different runs. Here, I do not use folds but the split function generates the training and a test set directly. The split ratio is a parameter defining the proportions of training and test set. For a classification task, independence can be verified easily by assuring that the split ratio is respected for all classes. This property can be utilized by the algorithm to create the split for the whole data set by first dividing it into one section per class, then splitting each section respecting the split ratio, and finally merging the section splits. If all sections can be split without a remainder, then the resulting overall split achieves independence perfectly. Otherwise, the best possible split can be found by rounding accordingly. Note that the imbalance caused by rounding gets stronger when the split ratio is badly chosen or when the classes are unbalanced.

When several splits are needed, this procedure can be repeated. Independence can be achieved by shuffling the sections before they are merged. The rounding should then consider the remainders to create corresponding probabilities, which should be used for every split independently to determine whether to round up or down.

For this data set group dependency has to be considered, too. Here, the two sessions of a smoker build a group that must not be split. Although group dependency seems like a common extension of the described procedure above, I did not find a machine learning framework, which offered it. The reason is that a *general* solution with arbitrary groups, and even approximations are NP-hard, as explained in Section 2.3.2. Sklearn [113], a common data analysis framework in python, names functions that consider associated subgroups of samples that may not be divided onto training and test set as *group*; for example GroupKFold or GroupShuffleSplit. These implementations do not consider stratification. Those who handle stratification, but no groups are called for example StratifiedKFold or StratifiedShuffleSplit.

However, I do not need a *general* solution, but one that can handle groups (in my case snippets which stem from the same subject), for which the same number is available. Thus, in my case subjects from the same class are interchangeable. For this case, I implemented a solution. Note that I recently found an implementation for another special case for sklearn [62], which allows only groups that belong to the same class. It is not yet part of the official release, and would however not suit my use case because my data contains groups, consisting of the two sessions of each smoker, that obviously span two classes: craving and non-craving.

In the following I do not present a general solution that allows for arbitrary groups, but only my implementation tailored for special case. The non-smoker class can be treated as before, because there are no additional restrictions. So this class can be split randomly. I treat craving and non-craving smokers identically by only choosing the split once, e.g., for the subjects of the craving class. Then I use the same split for the non-craving class.

Batch Generation

I only made few preliminary experiments on the batch size. Results did not vary significantly in the model quality, but only in the training time. The reasons for these differences are mostly in the processing on the GPU, where the overall time consumption consists mainly of two factors: transferring a batch to the GPU and processing the batch there. Batches must be small enough, so that the model parameters plus the values needed to process the batch fit into the the GPU memory. When batches are chosen too small, then time is wasted on overhead caused by the transitioning time between transferring data and processing data. My final choice was a batch size of 200 snippets.

With the stratification clarified I can now explain how I generate batches, which consider subject dependency and stratification. In order to minimize the variance induced by the batches, I make sure that the distribution within the batch differs as little as possible from the whole data set. The 200 snippets for one training batch stem from 49 measurements (see Table 5.1, p.106).

With $200/49 = 4, remainder = 4$, four measurements are used five times, the remaining ones four times. In my implementation I assure that from those four measurements 1 or 2 are from craving smokers, 1 or 2 are from non-craving smokers and 0 or 1 are non-smokers. To guarantee an minimal imbalance for several consecutive batches, the remaining measurements are sampled without replacement. Once all measurements are taken, they are all replaced. The implementation can be found in `classes/DataGenerator.py`.

Note that batch stratification is mainly needed for training. When testing, the model is fixed anyway, and thus the ordering of the testing samples can be arbitrarily chosen. To be able to easily analyze which snippets caused a prediction, I implemented another class, whose purpose is to generate only test data: `classes/TestdataGenerator.py`.

Transformation	snippet length	Algorithm	mean balanced acc.
none	several	Naïve bayes	≈ 49%
Fourier	several	Naïve bayes	≈ 60%
none	several	SVM	≈ 53%
Fourier	several	SVM	≈ 53%
Fourier	1,000	CNN	59%
none	400	MLP	59%
none	1,000	SVM	53%
Fourier	500	Naïve bayes	60%
Fourier	2,000	Naïve bayes	62%
Fourier	5,000	Naïve bayes	60%
Fourier	500	SVM	54%
Fourier	500	KNN (k=5)	53%
Fourier	500	KNN (k=7)	50%
Fourier	500	DT	53%
Fourier	5,000	DT	50%

TABLE 5.3: Overview of Pätz's results [112]

5.3 Earlier Results on this Data Set

Results by Pätz

The earliest results on this dataset were performed by Cedrik Pätz in his master's thesis [112] in 2017. In all of his experiments, he applies a Leave-One-Out Cross-Validation. The experiments with three classes were unsuccessful, he states that he was "not able to correctly separate the non-craving smokers from the non-smokers". Afterwards, he only discusses the craving vs. non-craving task. Two of his research goals are the effect of a Fourier transformation and the influence of the snippet length, for which he tries 100, 150, 250, 500, 1,000, 2,000, 5,000, 10,000, 20,000. Unfortunately, he only gives two overviews: First, one in which he shows the mean accuracy split by the number of snippets, but aggregated for all his models. Second, an overview that separates Naïve bayes and SVM models for Fourier transformed and untransformed input, but aggregates over all snippet lengths.

In Table 5.3, I summarize his results, where "several" snippet lengths means that all runs of all snippet lengths were considered and their mean value is given — these values are not given explicitly, but only in a figure, so I only give approximate values. Note that for each experiment the best performing snippet lengths were chosen. He does not present the details for each performed experiment. I agree with his conclusion: K-nearest neighbor, decision trees and support vector machines show a bad performance with about 53% class-balanced accuracy. The best models have accuracies around 60%: the Naïve Bayes classifier in combination with Fourier transformed data, and models that use neural networks. There is only a minor difference in results

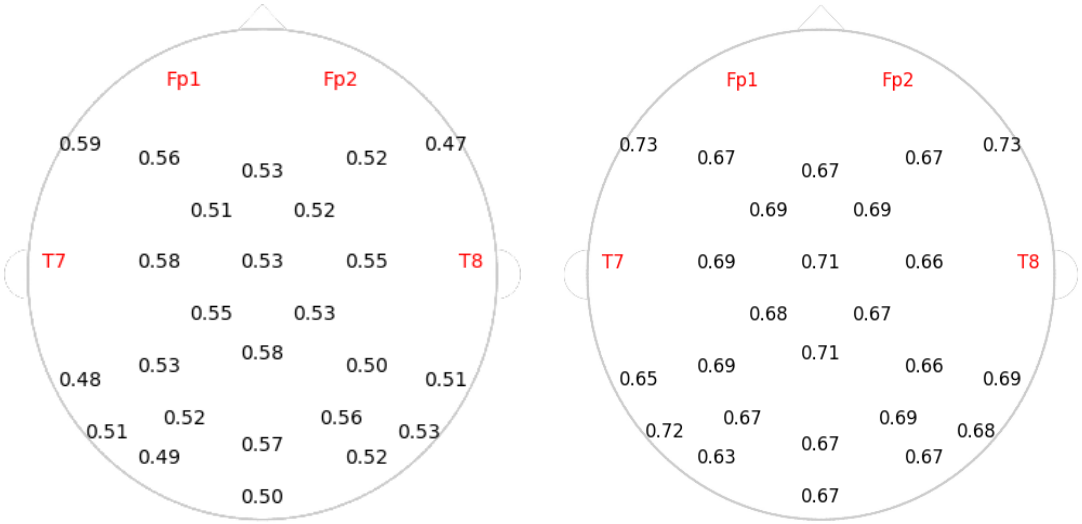


FIGURE 5.2: Channel-overview plot, left for a convolutional network, right for a dense network. (Source: [112])

between the convolutional network (CNN) and the one that uses only Dense layers (MLP).

In further experiments, Pätz is the first to investigate models that consider only data from one channel. He uses a window size 1000 and no transformation on the data. His plots show models performances as numbers near the location from which the data was taken. Unfortunately he does not explain which scores are shown, only that they "are not weighted prediction accuracies, but a higher score still means a higher prediction quality".

Interestingly, he finds that for single-channel models CNNs are strongly outperformed by networks that use dense layers. Further, he successfully reduces overfitting for dense networks by decreasing the number of parameters, and thus he improves the average score per channel from 66.52 to 68.24. Figure 5.2 shows the results for his best convolutional network (left), and the best dense network (right). Channel F8 (shown on the top right of each plot) is the worst channel of the CNN with a score of 0.47, but with 0.73 one of the best channels for the dense model. This, indicates that some important patterns are undetectable for the CNN, that can be found by the Dense network. One conclusion is that it cannot be said *in general* whether a channel is helpful to create a classification, but it also depends on the used model.

Finally, Pätz also compares predictions per subject. Sadly, he aggregates again over all performed experiments, so that clear conclusions are hard to be drawn. However, even with this strong aggregation it becomes clear that the results strongly depend on the subjects: all of the non-smokers have an average accuracy smaller than 50% while all smokers are above 50%. Although class weights are applied to all models that allow for it, many of the tested models use the smoker class as default prediction. For three non-smokers *bd53*, *dn20* and *vb79*, no model gives the correct prediction a probability of more than 80%. For *bd53* there is not even a single model that gives a pre-

diction probability of 15% or more. This person is predicted incorrectly by all models. I can only hypothesize that this person shows signs of addiction, maybe to another drug. More results on a subject specific level are presented in Section 5.4.2.

5.4 Own Experiments

In the following, I will answer the following questions. First I consider Multi-Channel models, then Single-Channel models:

Q1 (p. 24) What is the effect of preprocessing on the prediction quality?

Q2 (p. 44) Which loss function performs better? The classical combination of softmax and categorical cross-entropy or the cosine loss?

Q3 (p. 58) Is it sufficient to change the initialization to the identity function instead of using a skip connection?

Q4 (p. 61) Does enhancing convolutional blocks with skip connections improve the performance when performing EEG time series classification?

Q5 (p. 61) Do channel-wise 1D convolutions at the beginning of a model improve their performance?

Q6 (p. 61) Is it helpful to use skip connections in the channel-wise 1D convolutions?

Q7 (p. 67) Are there snippets, that are better suited than others? Can the class be seen in every snippet? What is the distribution of the snippet quality? Are there snippets that do not contain any helpful information? Are there snippets that mislead the classifier?

Q8 (p. 71) Which brain areas are best suited for the prediction of a given task?

Q9 (p. 86) Is there a bias towards the corners of the k-class projection simplex, when applying a neural network with a softmax layer to real data?

Q10 (p. 88) How do the different aggregations perform compared to each other? Is there a method that consistently outperforms the others?

Q11 (p. 88) How do other factors like the number of classes, the underlying distribution, the number of sub-objects, the fraction of information-carrying sub-objects and the strength of the bias towards the true class in these sub-objects influence the accuracy?

Q12 (p. 93) Can (linear) combinations of well performing aggregation functions preserve the best of both functions?

Q13 (p. 104) Can models distinguish between smokers and non-smokers?

Q14 (p. 104) Can models distinguish between craving smokers and non-craving smokers?

model	median	mean	variance
majority prediction	0.3333	0.3333	0.00000
random guessing	0.3333	0.3319	0.00094
Naïve Bayes	0.3112	0.3053	0.00771
Linear Discriminant Analysis	0.3143	0.3137	0.00078
LSTM network (466)	0.3457	0.3463	0.00447
neural network (453)	0.3769	0.3747	0.00558
neural network (453) (initialization)	—	—	0.00119

TABLE 5.4: Overview of model results for the three class problem

Q15 (p. 109) Which sampling technique performs better? A sliding window or a random sampling?

Before answering my own questions, I used the available source code of Pätz [112] to verify samples of his results. Thanks to his documentation, this was not a difficult process, but due to the high number of experiments and the long computation time, I only took few samples. The results were all in the expected range. By reviewing his code, it became clear that his single-channel models (from Figure 5.2) were supposed to show the mean class-balanced accuracy of five runs, but actually show the mean *unbalanced* accuracy due to a programming error. Figures respecting class-balance for the three-class problem will be given on page 123.

After this successful verification, I performed experiments with classical machine learning models that I missed in his experiments. Besides the fact that I use the samples from the non-craving smokers for my experiments, the data sets are identical. Further, I apply the following differing methods:

1. I use a shuffle split cross-validation with 100 runs instead of his Leave-One-Out; these results should cover the model space less locally and thus increase the reliability of the results.
2. Instead of the craving vs. non-smoker task, I try all combinations of two-class problems and the three class problem.
3. I always use a fixed snippet length of $1,024 = 2^{10}$; this allows pooling operations along the time axis by a factor of 2 for up to ten times, without the additional loss of information due to rounding (see Section 2.4.6).

5.4.1 Results of Multi-Channel Models

Table 5.4 gives an overview of the median, mean and variance values of the class-balanced accuracy for different models. Showing both median and mean gives an indication of the distributions skewness. Majority prediction constantly shows the same result. It always achieves 33.33% of class-balanced accuracy, independent of which class is predicted. Random guessing (with respect to class imbalance) shows a small variance, but mean and

median values close to $\frac{1}{3}$. Naïve Bayes and Linear Discriminant Analysis (LDA) perform even worse than random guessing. These models are unable to successfully generalize. I assume that this is caused by the given features. Without a separate feature detection they are not well suited for this task. Note that the LDA needs to be adapted to apply it to multivariate time-series. It can neither model the fact that what is considered to be independent variables are different points of the time series, nor can it handle different channels easily. In order to apply it, I reshaped the input, which hides the information that the data stem from different channels.

The only models that perform better than random guessing are based on neural networks, with the ability to create their own features automatically. I tested several networks based on Long Short-Term Memory (LSTM). I described this recurrent network structure earlier in Section 2.4.12. Some of the tested networks that use LSTM get time series as input directly, other variants apply convolutional layers as feature detectors, first. Most of the tested models apply a dense layer with softmax activation after the LSTM. All of these models reach at best the level of random guessing. The only model better than guessing is model *466*, which has a class-balanced accuracy of 34.63% and a t-test returns $p = 0.11$, thus this model performs *not* significantly better than guessing. Model *466* utilizes the input snippets directly and does *not* use a final softmax layer, but applies three LSTM neurons that are directly taken as output[1]. The dimensions are a batch-size of 200 with 1024 time-steps for 25 channels. The LSTM block is processed as described in Section 2.4.12. I use the standard options and return only the output vector of the last time step. Although LSTM was especially designed for time series, it seems not to be well suited for this task[2].

My best-performing classical neural network is model *453*. With about 37.5% accuracy and a p-value of $1.98 \cdot 10^{-6}$ returned by a t-test, the results are significantly better than random guessing. Thus, I am able to verify that there is a significant difference between the EEG signals of craving smokers, non-craving smokers and non-smokers. Note that all described convolutional network models apply convolutions without dilation (see Section 2.4.5). I have also tested models with dilated convolutions, but they yielded worse results.

The last row of Table 5.4 shows model *453* again, and describes the variance of 100 runs that were performed on the same split. Assuming that the obtained variance is a typical value for an arbitrary split, I can estimate the variance resulting from the initialization. So from the total variance $Var(S + W) = 0.00558$, weight initializations is measured to cause about $Var(W) = 0.00119$. Thus, the variance of the split can be estimated to be $Var(S) = 0.00558 - 0.00119 \approx 0.0044$. So the size of the variance created by the split is more than three times the size compared to the initialization. Note that I omit median and mean values for the network initialization in Table 5.4 to prevent

[1] In model *467*, I added a softmax activation after the LSTM. It returns values worse than random guessing.

[2] Source code for all models is given under `models/*`. Models using LSTM start with `46`.

confusion with models run on different splits. Using only one split results in mean and median values that are bad indicators of the model quality because they show strong variations depending on the split.

Figure 5.3 shows the structure of model *453r*, which is the residual version of model *453*. The plain version is my best performing multi-channel neural network model[3]. It has only few layers, uses only few convolutions, no pooling and most neurons are in the dense layer with 1024 neurons.

Although it does use convolutions, it is more of a dense network. To clarify the most complex version, I depict the residual version — meaning with channel-wise convolution and skip connections. The convolutional version does not have skip connections; the plain version does also not have the channel-wise 1D convolution. The remainder is identical in all versions.

The following list emphasizes the network's layers, not just for the most complex, but for all variants of the model. The split in layers 5 and 6 has similarities with residual connections. This idea is not new. In some implementations (e.g., by Dietz [31]) instead of a residual, a convoluted residual is forwarded on to the next layer.

1) Input layer: 25 channels with 1024 time steps
2) (optional) channel-wise 1D convolution
3) (optional) skip connection around channel-wise 1D convolution
4) Merge layer
5up) 1D convolution (kernel size: 1, filter size: 1)
5down) Flatten
6up) Flatten
6down) Dense layer: 1024 neurons
7) Add(6up, 6down)
8) Dense layer: 3 neurons (Softmax)

[3]The results described in my earlier work [35] are better, but are incorrectly labeled. They do not stem from a multi-channel experiment but use data from only one channel: Fz.

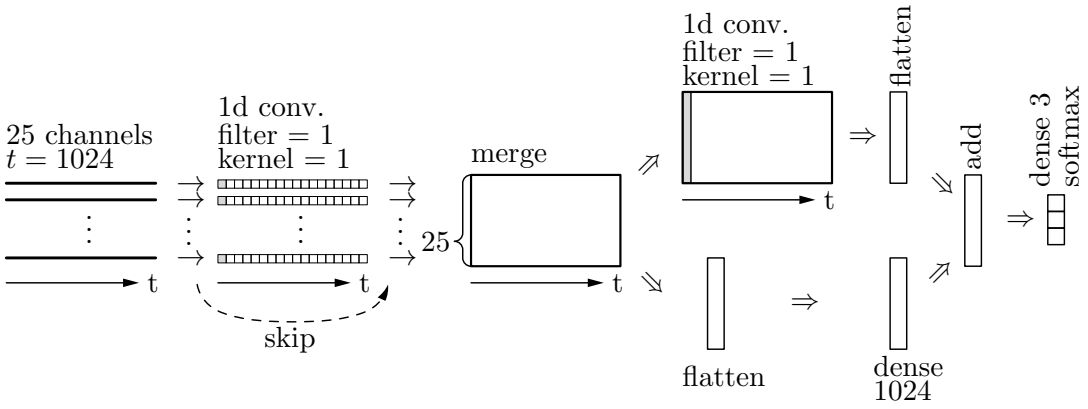


FIGURE 5.3: Network structure of model *453r*

- Optimizer: Adadelta[4]
- Learning rate: 1.0 (default)
- Loss function: categorical cross-entropy
- Number of epochs: 100
- Number of batches per epoch: 50
- Batch size: 200
- Total number of parameters: 26,218,575

Multi-Channel Results with Aggregation

So far, for testing, one batch of randomly chosen 200 snippets was taken from the testing set, similar as in training, which is why I name it *like training*. This corresponds only to at total of approximately 6.7 minutes (see Section 5.1) although 9.5 minutes are available for *each* measurement. Thus, I follow the more realistic approach, in the following, which is also applied for example in brain computer interfaces: I use several snippets from the same measurement and aggregate their results. Details on aggregation are given in Section 2.5. Note that showing the median and mean values of the models returned by the 100 runs would put higher weights to those subjects that were in the test set more frequently. To avoid this imbalance, I show instead values that would be returned by an average model. Thus I return the average and mean values computed for each measurement weighted with the corresponding class weight. The source code for this computation is given in `ResultEvaluator.py`.

Table 5.5 shows the performance of all three versions of model *453* on multi-channel input. Plain and without aggregation, it achieves about 37.6% accuracy, as already seen in Table 5.4. Here, I additionally show the effects of different aggregation methods, for both sampling schemes and the three model versions 'Plain', 'Conv.' and 'Res'. Nearly all median and mean values are higher when any aggregation is performed, for both sampling schemes. So it can be stated that any aggregation is better than no aggregation. Generally one effect of aggregation is to intensify the skewness of the distribution away from random guessing. I assume that the lower median value for random sampling and *prod* is an outlier.

> **Answer** (**Q15**, p. 109): The sampling method shows only minor differences. It seems like sliding window performs on average slightly better than random sampling. One reason might be a slightly better generalization due to a higher guaranteed coverage of the training samples.

[4] Adadelta is the optimizer I used in most of the tried models, as it showed better results than stochastic gradient descent and similar results as the Adam optimizer in previous experiments. Results of this model that used the Adam optimizer are given in Appendix B.1 (p. 181).

		Plain		Conv.		Res.	
sampling	agg.	median	mean	median	mean	median	mean
like training	None	0.3769	0.3747	0.3540	0.3632	0.3535	0.3588
random	dvote	0.3888	0.3955	0.3888	0.3844	0.3333	0.3655
random	psum	0.3888	0.3916	0.3888	0.3844	0.3333	0.3655
random	qsum	0.3888	0.3938	0.3888	0.3866	0.3333	0.3650
random	pmax	0.3888	0.4038	0.3888	0.3966	0.3888	0.3811
random	avote1	0.3888	0.3938	0.3611	0.3805	0.3333	0.3661
random	prod	0.3333	0.3894	0.3611	0.3850	0.3333	0.3700
s. window	dvote	0.3888	0.3994	0.3888	0.3805	0.3333	0.3650
s. window	psum	0.3888	0.3955	0.3888	0.3833	0.3333	0.3650
s. window	qsum	0.3888	0.3983	0.3888	0.3805	0.3333	0.3666
s. window	pmax	0.4166	0.4166	0.3888	0.3950	0.3888	0.3972
s. window	avote1	0.3888	0.3977	0.3888	0.3816	0.3333	0.3644
s. window	prod	0.3888	0.3938	0.3888	0.3855	0.3333	0.3661

TABLE 5.5: Result overview for the best Multi-Channel neural network model *453* for both sampling schemes, the basic aggregation functions and three variants 'Plain', 'Conv.' and 'Res' for the early layers of the models

For the three model variations it becomes clear that the convolutional and residual versions perform clearly worse than their plain counterparts. Due to the few participants I did not use a validation set. Thus it is neither possible to prevent overfitting by early stopping before iteration 100, nor to reduce the learning speed on plateaus. In the data set described in Chapter 6, there are sufficiently many samples for a validation set and therefore I can and do apply these techniques. This is successful, when I combine them with a generally reduced learning rate for convolutional and residual models, there. As reducing the general learning rate does not need a validation set, it is possible here too. However, experiments with a reduced learning rate show worse results, here. These models are unable to adapt to the training samples properly and do not surpass the level of random guessing. So the results for 'Plain', 'Conv.' and 'Res.' are generated using the default learning rate 1.

> **Answer**(**Q5,Q6**, p. 61): Both, channel-wise 1D convolutions at the beginning and additional skip connections decrease the model performance, for multi-channel models on the smoker data set.

Questions **Q3** (p. 58) and **Q4** (p. 61) are based on the assumption that the network uses several convolutional or residual layers. As this is not the case for the described model, they cannot be answered, here. However, models that make use of convolutional or residual blocks returned worse results.

> **Answer** (**Q10**, p. 88): For the multi-channel experiments, there is a clear best aggregation function: *pmax*. For all three versions of the model and both sampling methods, it is the winning function. For the plain model and sliding window sampling, the median value reaches even 41.66%. The other aggregation functions perform similarly. There is no clear order between them.

		Plain		Conv.		Res.	
sampling	agg.	median	mean	median	mean	median	mean
like training	None	0.3769	0.3747	0.3540	0.3632	0.3535	0.3588
random	ppmix1	0.3611	0.3916	0.3611	0.3866	0.3333	0.3677
random	ppmix2	0.3888	0.3911	0.3611	0.3866	0.3333	0.3677
random	pqmix1	0.3888	0.3922	0.3611	0.3900	0.3333	0.3683
random	pqmix2	0.3888	0.3916	0.3888	0.3922	0.3611	0.3705
random	fano1	0.3888	0.3916	0.3888	0.3816	0.3333	0.3672
random	fano2	0.3888	0.3922	0.3888	0.3811	0.3333	0.3672
s. window	ppmix1	0.3888	0.3961	0.3888	0.3883	0.3333	0.3666
s. window	ppmix2	0.3888	0.3961	0.3888	0.3877	0.3333	0.3655
s. window	pqmix1	0.3888	0.3950	0.3888	0.3872	0.3333	0.3733
s. window	pqmix2	0.3888	0.3944	0.3888	0.3850	0.3333	0.3683
s. window	fano1	0.3888	0.3966	0.3888	0.3833	0.3333	0.3683
s. window	fano2	0.3888	0.3966	0.3888	0.3827	0.3333	0.3683

TABLE 5.6: Result overview for the best multichannel neural network model for both sampling schemes, the advanced aggregation functions and three variants for the early layers of the modelsindividual results, on the real data for multi-channel models. The *pmax* function is the best aggregation function in that case.

This result is surprising, as *pmax* was the function that was by far the worst in the performed simulations in Chapter 4. This strengthens the hypothesis that the underlying distribution of the real data here differs significantly from the ones chosen in the simulation.

Table 5.6 gives the results for model *453* for those aggregation functions that combine *pmax* and *prod*. As both performed not extraordinarily well, it does not surprise that their combination reaches a similar level.

> **Answer** (**Q12**, p. 93): Linear combinations of *prod* and *psum* cannot outperform their individual results, here. The *pmax* function remains the best aggregation function for this data set.

Results of the Four Different Tasks

Table 5.7 shows the results for the models that consider only two classes. Again, I consider both sampling schemes and the basic six aggregation functions. The right two columns consider craving vs. non-smokers, as Pätz [112]. His 59% accuracy without aggregation with Leave-One-Out cross-validation and thus looking only to a very local area in the model space seems to be optimistic. With random-shuffle split, I reach only 57%. Only with applied aggregations, my models can outperform the old ones, reaching 60%. The highest value is generated by *pmax* aggregation and sliding window sampling, 61.75%.

The results for craving vs. non-craving show a mean value of 54.23% and are thus worse than craving vs. non-smoker. This was expected, as the assumed

		c vs. nc		nc vs. ns		c vs. ns	
sampling	agg.	median	mean	median	mean	median	mean
like training	None	0.5525	0.5423	0.4983	0.5032	0.5716	0.5737
random	dvote	0.5833	0.5558	0.5000	0.5191	0.5833	0.6016
random	psum	0.5833	0.5575	0.5000	0.5225	0.5833	0.6025
random	qsum	0.5833	0.5575	0.5000	0.5225	0.5833	0.6025
random	pmax	0.5416	0.5533	0.5833	0.5450	0.5833	0.5841
random	avote1	0.5833	0.5566	0.5000	0.5200	0.5833	0.6008
random	prod	0.5833	0.5500	0.5000	0.5300	0.5833	0.5916
s. window	dvote	0.5833	0.5491	0.5000	0.5200	0.5833	0.5983
s. window	psum	0.5833	0.5533	0.5000	0.5191	0.5833	0.5975
s. window	qsum	0.5833	0.5533	0.5000	0.5191	0.5833	0.5975
s. window	pmax	0.5833	0.5725	0.5833	0.5766	0.6666	0.6175
s. window	avote1	0.5833	0.5425	0.5000	0.5200	0.5833	0.5966
s. window	prod	0.5833	0.5500	0.5000	0.5191	0.5833	0.5941

TABLE 5.7: Result overview for the best multichannel neural network model for both sampling schemes, the basic aggregation functions and the three 2-class tasks

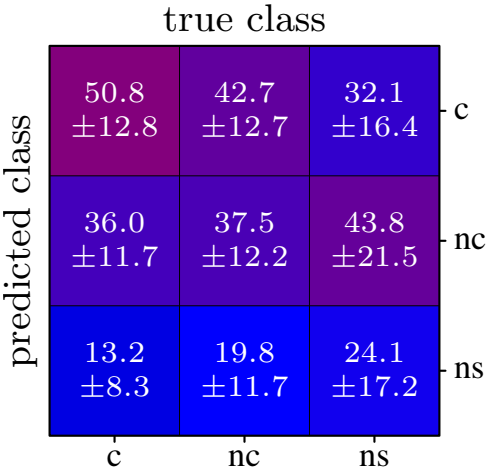


FIGURE 5.4: Confusion matrix of model *453* without aggregation

effects of craving and smoking are combined, here. The *t*-test returned a *p*-value of $1.25 \cdot 10^{-6}$ which verifies the significance of the *craving* effect.

The model has however severe problems to distinguish non-craving from non-smokers. Without aggregations, the values are only at the level of random guessing. With aggregations, they also reach only 52% to 53%. The only exception is again *pmax*, for which 54.5% and 57.66% are reached for random sampling and sliding window. This result is especially interesting since here: although the average value is at about 50% without aggregation, the additional information that several snippets stem from the same measurement considerably helps the model.

Figure 5.4 depicts the confusion matrix of model *453* in a version showing the mean value and standard deviation from the 100 runs. The entries contain the average normalized values and their variance as numbers and more reddish color indicates a higher mean value. Non-smokers are correctly predicted in

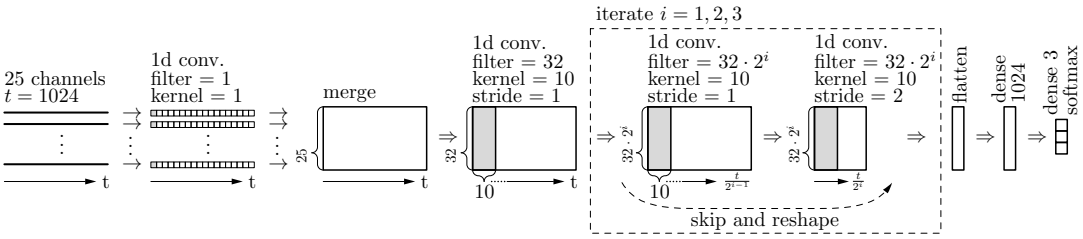


FIGURE 5.5: Network structure of the best single-channel model: *423N*

only 24% of the cases and are the worst-predictable class with non-craving being second best and craving being best-predictable. The matrix confirms the findings from the 2-class models: It is extremely difficult for the model to distinguish non-craving from non-smoker. This value is with 43.8% the worst of the incorrect predictions. It also shows with 21.5 the highest variance. The second worst error occurs when the model predicts craving although the subject is non-craving. Craving and non-smoker are confused most rarely, which again confirms the results of the 2-class models.

> **Answer** (**Q13**, p. 104): Model *453* is able to successfully distinguish between smokers and non-smokers significantly better than random guessing. If the smokers are not craving, then the models need to aggregate several snippets from the same person (best with max aggregation) in order to reach a significant level.
>
> **Answer** (**Q14**, p. 104): Model *453* is able to successfully distinguish craving and non-craving smokers better than random guessing. Even without aggregation, the model reaches a significant level.

5.4.2 Results of Single-Channel Models

In the following, I only show the results of few variants of one basis model, this time for one single channel instead of multiple channels as model input. Although it is possible to optimize the model structure independently for each channel, I decided to use the same model structure for all channels. The one that (on average) performed best, is model *423N*, for which the network structure is depicted in Figure 5.5. The picture shows that, again, snippets of length t = 1024 are used, but also shows the input layer with 25 channels. This may be confusing, but is caused by the fact that I use identical models for both cases, multi- and single-channel. This enables a better comparability especially in the number of used weights. In the single-channel case, the data from one channel is copied 25 times and used as model input. The following 1D convolution can rescale each channel, before it is merged. Then follows one convolutional block with 32 filters and a kernel size of 10. Afterwards, I use three residual blocks, each consisting of two 1D convolutions for which in each iteration the number of filters is doubled and the time dimension is halved. So in total the numbers of neurons before a block and after a block are identical. I use a stride of 2 in the second convolution to reduce the time axis, here. In other versions of the models I apply MaxPooling, instead —

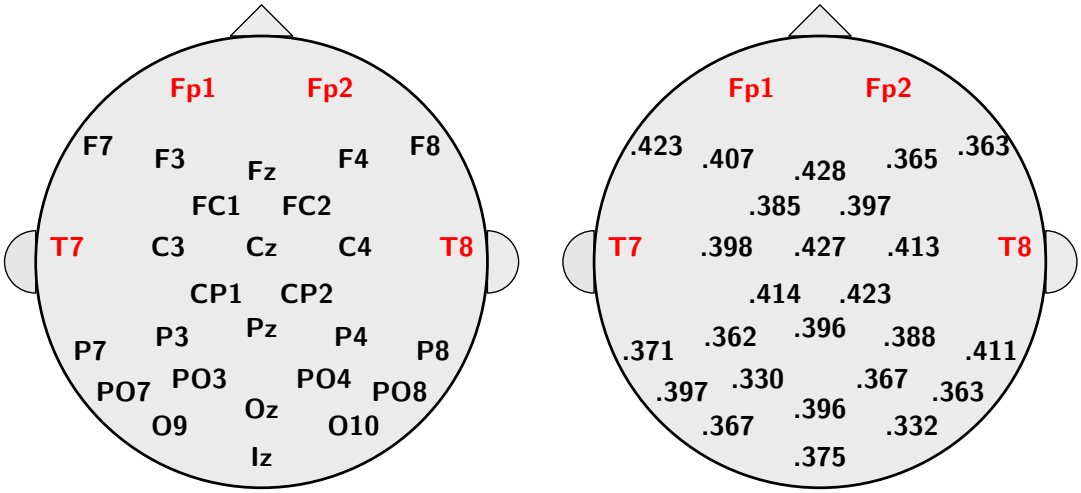


FIGURE 5.6: Headplot of model *423N* (conv.) showing the mean values from shifted window sampling and *avote1* aggregation (right)

with similar results. With the same number of neurons before and after a block, the skip connection retains all information. However, because of the changed dimensions, it needs a reshape. As typical for convolutional networks, the beginning part just described serves as feature detector, which is separated by a flatten layer from the part where the features are combined into a prediction. There, I use 1024 neurons in the dense layer and finally perform the classification using softmax.

- Optimizer: Adadelta
- Learning rate: 1.0 (default)
- Loss function: categorical cross-entropy
- Number of epochs: 100
- Number of batches per epoch: 50
- Batch size: 200
- Total number of parameters: 33,894,741

Figure 5.6 gives the result overview for model *423N* with 1D convolution using shifted window sampling and *avote1* as aggregation. The channels marked in red were omitted because they contained too much noise that could not be removed. The left plot is given as reminder of the channel names — on the right the names are replaced by the corresponding result mean values. Many channels show results over 40% and are thus even better than multi-channel models. Channel Fz is the best channel and reaches 42.8%, while PO3 is even worse than random guessing. Also, if both were considered outliers, most of the channels surpass the level of random guessing. The head's central region with channels Cz, CP1, CP2, but also C3, C4 seem to be especially well suited to detect signs of smoking or craving. Also models using data from the frontal left region (channels F7 and F3) show good results.

	Plain		Conv.		Res.	
channel	median	mean	median	mean	median	mean
C3	0.3888	0.3955	0.3888	0.3977	0.3888	0.3972
C4	0.3888	0.4061	0.3888	0.4127	0.3888	0.4022
CP1	0.3888	0.4088	0.3888	0.4138	0.3888	0.4088
CP2	0.3888	0.4238	0.4444	0.4227	0.3888	0.4161
Cz	0.3888	0.4233	0.3888	0.4266	0.3888	0.4283
F3	0.3888	0.3983	0.3888	0.4066	0.4444	0.4200
F4	0.3333	0.3633	0.3333	0.3650	0.3333	0.3677
F7	0.4444	0.4338	0.4444	0.4233	0.4444	0.4205
F8	0.3333	0.3627	0.3333	0.3633	0.3333	0.3638
FC1	0.3888	0.3777	0.3888	0.3850	0.3611	0.3783
FC2	0.3888	0.3922	0.3888	0.3966	0.3888	0.3855
Fz	0.4444	0.4533	0.4444	0.4283	0.4444	0.4450
Iz	0.3888	0.3827	0.3333	0.3750	0.3611	0.3683
O10	0.3333	0.3300	0.3333	0.3316	0.3333	0.3344
O9	0.3333	0.3572	0.3888	0.3666	0.3888	0.3611
Oz	0.3888	0.3833	0.3888	0.3955	0.3888	0.4022
P3	0.3333	0.3461	0.3333	0.3616	0.3333	0.3494
P4	0.3888	0.3850	0.3888	0.3883	0.3888	0.3994
P7	0.3888	0.3744	0.3333	0.3711	0.3888	0.3744
P8	0.4444	0.4205	0.3888	0.4105	0.3888	0.4000
PO3	0.3333	0.3150	0.3333	0.3300	0.3333	0.3216
PO4	0.3333	0.3522	0.3888	0.3666	0.3888	0.3738
PO7	0.3888	0.3922	0.3888	0.3966	0.3888	0.3927
PO8	0.3333	0.3577	0.3333	0.3633	0.3888	0.3650
Pz	0.3888	0.3994	0.3888	0.3961	0.3888	0.3966
sum	9.4428	9.6345	9.4428	9.6944	9.5539	9.6723

TABLE 5.8: Result overview for the best single-channel model *423N* on channel Fz, for shifted window sampling, avote1 aggregation and three variants 'Plain', 'Conv.' and 'Res' for the early layers of the models

> **Answer** (**Q8**, p. 71): The best brain area for distinguishing craving smokers, non-craving smokers and non-smokers seems to be the central area of the head. The best model used data from the frontal central (Fz) channel, but also models which use data from the frontal left area (F7, F3) perform well.

Table 5.8 gives a similar overview as Figure 5.6. In the table, it is difficult to visualize the channel's locations on the scalp, but it enables an easy comparison between the three model variants 'Plain' (without the channel-wise 1D convolution at the beginning of the model), 'Conv.' (with the channel-wise 1D convolution) and 'Res.' (with the convolution and a corresponding skip connection). Table 5.8 illustrates the results for each channel: The overall conclusion is that all three versions perform similarly for most of the channels. If a channel performs well for one version, it is likely to perform well for the other versions as well. For example channel *Fz* is the best channel for all versions and channel *PO3* is the worst.

Because the median can only take few values, the differences are often zero. I consider the sum of the mean values and see that with a sum of 9.6944, the convolutional versions performs indeed slightly better than *Res* with 9.6723 and *Plain* with 9.6345. However, these differences are very small and none of the performed pairwise *t*-tests showed significance.

> **Answer**(**Q5,Q6**, p. 61): For the single-channel case the channel-wise 1D convolution performs on average slightly better than models without it or models with an additional skip connection. However, the effect is not significant.

Combined Aggregation Functions for Single-Channel Models

Table 5.9 shows the mean values of the results obtained for the combined aggregation functions, *sum* and *prod* as well as for no aggregation for each channel. For each row I marked (all) maximal values by using bold font. The differences between the aggregation functions is generally small, and even smaller for such functions that differ only in a parameter (*ppmix*, *pqmix* and *fano* each with parameter set 1 and 2. For a better overview, I added one row to count the channels for which this function was best, one row which shows the average value of all channels and finally one row that shows the average rank the function yields. Note that small values lead to high ranks and thus high values in this row indicate good performance.

For the average value, the worst function is *prod*, although it achieves maximal values for 7 of 25 channels. The *psum* function is in 9 of 25 channels the optimal choice and also has the highest average value as well as an average rank of more than 6. Function families *ppmix* and *pqmix* are bad choices in all three criteria. They are outperformed by both versions of *fano*. Interestingly, when regarding only the average rank, the *fano* family outperforms even *psum*. Overall, *fano2* is always better than *fano1*.

> **Answer** (**Q12**, p. 93): For each of the tested combined aggregation functions, there is a channel for which it results in the maximal value reached. However, on average *psum* returns the highest value in most of the cases and also the highest average value, which makes it a solid choice for single-channel models. From the tested combined aggregation functions the *fano* family outperforms *ppmix* and *pqmix*.

The best single-channel Model: Fz

Table 5.10 gives the results for the best single-channel model on channel Fz. With the gained information I answer the questions on the sampling method and the best-performing aggregation function for single-channel models.

> **Answer** (**Q15**, p. 109): For the best single-channel model, the sliding window and random sampling approaches perform similarly.

ch.	None	psum	prod	ppmix1	ppmix2	pqmix1	pqmix2	fano1	fano2
C3	.3795	.4044	**.4111**	.4027	.4033	.4038	.4033	.4033	.4033
C4	.3698	**.4161**	.4038	.4094	.4127	.4094	.4077	.4094	.4100
CP1	.3807	.4216	.4227	**.4261**	.4255	.4205	.4222	.4222	.4222
CP2	.3878	.4288	.4294	.4288	.4288	**.4305**	**.4305**	.4288	**.4305**
Cz	.3915	**.4255**	.4205	.4222	.4205	.4238	.4211	.4250	.4250
F3	.3728	**.4016**	.3733	.3888	.3866	.3866	.3961	.3983	.3983
F4	.3461	.3694	**.3711**	.3655	.3650	.3677	.3672	.3683	.3672
F7	.3810	**.4044**	.3944	.3977	**.4044**	.4000	**.4044**	.4027	.4027
F8	.3542	.3605	**.3711**	.3644	.3666	.3661	.3638	.3605	.3605
FC1	.3626	**.3894**	.3811	.3866	.3888	.3877	**.3894**	**.3894**	**.3894**
FC2	.3743	.3983	**.4027**	.4000	.3972	.3972	.3994	.4000	.4011
Fz	.3971	.4261	.4194	.4227	.4222	.4227	.4216	**.4266**	**.4266**
Iz	.3569	.3738	.3700	**.3777**	.3772	.3738	.3738	.3738	.3738
O10	**.3383**	.3316	.3277	.3305	.3300	.3272	.3283	.3322	.3322
O9	.3315	.3605	.3572	.3572	.3605	.3577	**.3611**	**.3611**	**.3611**
Oz	.3660	.3922	**.3944**	.3916	.3894	.3905	.3905	.3922	.3922
P3	.3533	.3644	.3605	.3655	.3650	.3650	.3594	**.3666**	.3655
P4	.3695	.3888	.3838	.3888	.3883	.3855	.3861	**.3905**	**.3905**
P7	.3442	**.3694**	.3633	.3650	.3644	.3655	.3650	.3688	**.3694**
P8	.3808	**.4077**	.4005	.3994	.3983	.3950	.4011	.4033	.4050
PO3	.3194	**.3288**	.3150	.3283	.3266	.3238	.3255	.3283	.3283
PO4	.3623	**.3733**	.3688	.3711	.3694	.3688	.3722	.3722	.3722
PO7	.3536	.4016	**.4088**	.4061	.4077	.4066	.4061	.4011	.4016
PO8	.3506	.3600	.3533	**.3638**	.3627	.3600	.3600	.3611	.3611
Pz	.3760	.3944	**.4061**	.3950	.3955	.3944	.3933	.3972	.3972
avg.	.3640	.3877	.3844	.3862	.3863	.3852	.3860	.3873	.3875
#	1	9	7	3	1	1	4	5	6
rank	1.36	6.38	4.5	5.28	5.06	4.42	4.88	6.4	6.72

TABLE 5.9: Mean result values for several (combined) aggregation functions for all channels of single-channel models and sliding window sampling

As for single-channel, applying aggregation improves the results. This holds for all tested functions except for *pmax*, whose mean values are a bit smaller than without aggregation.

> **Answer** (**Q10**, p. 88): For the single-channel experiments, there is no clear best aggregation function, but *pmax*, which was outstandingly the best in the multi-channel case is the worst here. Interestingly, one finding from multi-channel still holds: The other aggregation functions perform similarly, with *prod* performing slightly different: for sliding window worse, for random sampling a bit better than the others. Besides that, there is no clear order between the aggregation functions.

In Figure 5.7, I show the confusion matrix of model *423N* for channel Fz — again without aggregation. This is the overall best model for this data set. Again, non-smoker is the class that is wrongly predicted most often. But compared to other models, the correct classifications have strongly increased for non-smokers. Also, the correct detection rate for non-craving smokers has

sampling	agg.	median	mean
like training	None	0.3925	0.3971
random	dvote	0.4444	0.4311
random	psum	0.4444	0.4305
random	qsum	0.4444	0.4266
random	pmax	0.3888	0.3850
random	avote1	0.4166	0.4277
random	prod	0.4444	0.4333
s. window	dvote	0.4444	0.4300
s. window	psum	0.4444	0.4261
s. window	qsum	0.4444	0.4305
s. window	pmax	0.3888	0.3866
s. window	avote1	0.4444	0.4288
s. window	prod	0.4444	0.4194

TABLE 5.10: Result overview for the best single-channel model *423N* on channel Fz, for both sampling schemes, the basic aggregation functions

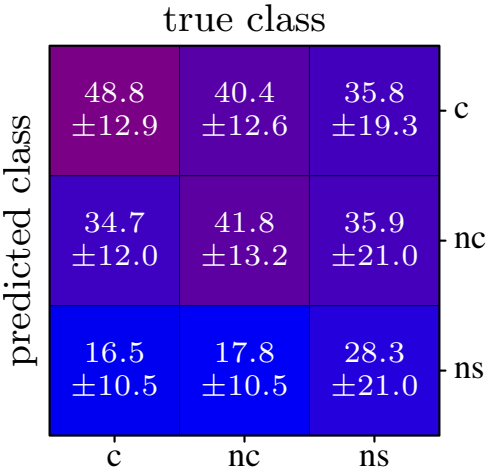


FIGURE 5.7: Confusion matrix of the single-channel model *423N* for channel *Fz*

increased to 41.8%. The strongest sources of confusion still occur between craving and non-craving. The number of actual non-smokers classified to be non-craving is smaller than for any of the multi-channel models. Craving is still the class with the highest probability to be predicted correctly. However, compared to other models, craving smokers are incorrectly predicted more frequently, here.

Hyperparameter Optimization

Using no validation set has the disadvantage that model and hyperparameter tuning (e.g., adapting the model structure, learning rate or optimizer) can only be performed on the test set. With this procedure, there is neither the possibility to *examine* overfitting of the *parameters* nor to perform early stopping to *prevent* this type of overfitting. Further, iteratively adapting the models with regard to the results on the test set bears the risk of overfitting

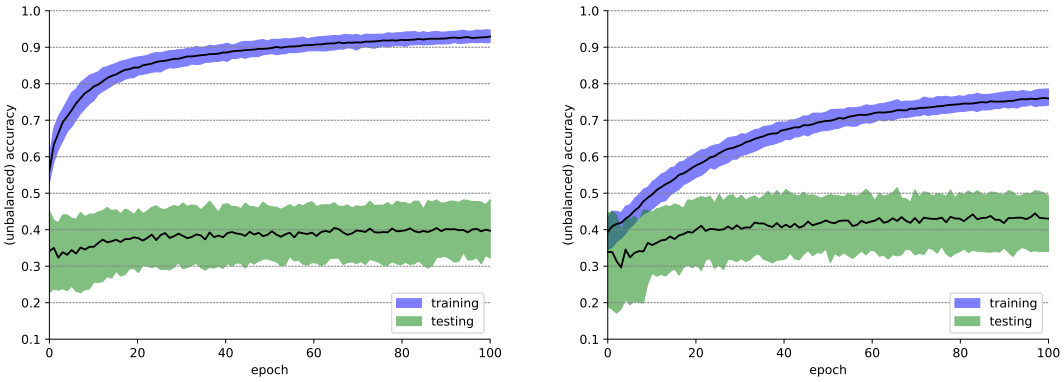


FIGURE 5.8: Training accuracy histograms: left for model *453*, right for *423* Fz

the *hyperparameters*. Still, hyperparameters and model structures need to be determined somehow.

Strictly mathematically speaking this is not allowed, because once a model was trained with a fixed set of hyperparameters and tested on the test set, this result may not be used in order to find a better model, as it might overfit. But the only way to find better models and hyperparameters is to try several of them and choose the most promising ones, as I do in this work. I think this is the most severe issue with the given data set and my processing in this work. However, with so few data samples, I think this is the only viable option.

Adjusting the hyperparameters can cause two types of overfitting: First, regarding to some test sets or second, regarding to the entire data set. There is no way to distinguish between these two. However, with 100 different splits and thus also 100 different test sets it is possible but unlikely that a a set of hyperparameters works well only on these specific test sets. It is more likely that the hyperparameters are overfitting to the whole data set. This means that the hyperparameters work for the given data set, but would not work on a new, independent set. And as consequence, I strongly recommend to verify all results obtained from this data set to be tested again, with another independent data set.

To be able to verify the amount of overfitting, I save the score obtained during training for the training and the test set. Unfortunately, the Keras framework does not support class weights during training. Thus, I show the unbalanced accuracy as y-axis in Figure 5.8 for model *453* on the left and for model 423 for channel Fz on the right. The x-axis shows the 100 epochs performed — but, as I can create a huge amount of different snippets, I define an epoch to consist of 50 training batches (with a batch-size of 200). The black lines indicate the mean values, the blue and green bands indicate the area between the upper and lower decile. Thus, the colored area shows the range of the middle 80 models.

One can observe several effects: First, the range is much smaller for training than for testing. Second, although the y-axis shows the *unbalanced* accuracy,

		423N Fz		*453*	
sampling	agg.	median	mean	median	mean
like training	None	0.3574	0.3666	0.3476	0.3612
random	dvote	0.3611	0.3833	0.3333	0.3483
random	psum	0.3888	0.3822	0.3333	0.3500
random	qsum	0.3888	0.3833	0.3333	0.3550
random	pmax	0.3333	0.3405	0.3333	0.3700
random	avote1	0.3611	0.3794	0.3333	0.3494
random	prod	0.3888	0.3900	0.3333	0.3488
s. window	dvote	0.3888	0.4022	0.3333	0.3500
s. window	psum	0.3888	0.4038	0.3333	0.3483
s. window	qsum	0.3888	0.4038	0.3333	0.3494
s. window	pmax	0.3333	0.3422	0.3333	0.3522
s. window	avote1	0.3888	0.4061	0.3333	0.3538
s. window	prod	0.3888	0.4072	0.3333	0.3427

TABLE 5.11: Predictions of models *423N* on channel Fz, and *453* for both sampling schemes and the basic aggregation functions when run on non-preprocessed data

the models are created to maximize the *balanced* accuracy. Therefore the plots do not show naive predictions at the level of guessing the majority class, which would yield an unbalanced accuracy score of $\frac{4}{7} \approx 42.86\%$. Third, is an effect of overfitting: The accuracy on the training set is increasing faster and more consistently than on the test set. Although this difference is increasing for later epochs, it is recommended to keep training, as long as the accuracy on the test set is increasing. At the end of the training, the multi-channel model reaches reaches a higher mean value (93%) on the training and a lower value (40%) on the test set. The single-channel shows 76% and 43% respectively. Thus, I conclude that the multi-channel model shows a stronger overfitting. This is plausible, because of the higher input dimension in that case. Note that the unbalanced accuracy relates very well to the balanced values *like training* from Table 5.5 and Table 5.10.

Evaluation on Non-Preprocessed Data

Preprocessing removes unwanted noise, but the effect on the prediction quality is unknown, because this noise might be helpful or distracting for the classifier. Thus, I perform experiments, in which I reduced the preprocessing to a minimum: First, I removed the three participants whose signals were dominated by noise and second I removed the noisy channels Fp1, Fp2, T7 and T8. No frequency filtering is applied and also no independent component analysis. I compare the results generated from this data set with the one where additional filtering and ICA is applied.

The results are given in Table 5.11 for the best models for multi-channel and single-channel data. Note that this comparison is not completely fair in the sense that I used much effort to optimize the hyperparameters of the models on the preprocessed data and I use the same hyperparameters here.

Both models reach mean values above 36% when tested *like training* and are both significantly better than guessing. But they are also both worse than their counterparts on the preprocessed data that show mean values of 0.4064 and 0.3747 (single- and multi-channel). The significance of this effect can only be confirmed for model *423N* ($p = 0.0018$), but not for the multi-channel case ($p = 0.1837$).

With applied aggregation, it becomes clearer that the single-channel model outperforms its multi-channel counterpart. Median values of the latter model are consistently at 33.33% — the level of random guessing. On the other hand, model *423N* is with the exception of *pmax* always better than guessing. As seen before for single-channel, the performance with aggregation is higher than without. The difference between the two sampling methods here is bigger than in the earlier experiments. Sliding window is often more than 2% better than random sampling. With this sampling scheme, *prod* reaches the highest value: 40.72%. Again, the sum-based and vote-based aggregation functions perform similarly — approximately 40.4% for sliding window and 38.1% for random sampling.

The multi-channel model shows an effect, which has not showed up, before: results for all aggregation functions but *pmax* are worse than the original model. I suppose that the stronger noise on multiple channels prevents the model from generalizing properly.

> **Answer** (**Q1**, p. 24): Models perform better on preprocessed data. This effect is significant for single-channel models but is not significant for multi-channel models. The model run on non-preprocessed multi-channel data shows a special side-effect: it performs worse, when aggregation is applied.

Evaluation of Cosine Loss

Barz et al. [7] recommend to use the cosine loss, instead of the classical combination softmax and cross-entropy loss for small data sets. They showed for pictures that this returns better result, when applied on small data sets. Thus, I also performed experiments on the smoker data set and show the results in Table 5.12.

Note that the results should be seen in comparison to the ones obtained with classical softmax and cross-entropy loss, given in Table 5.5 for the corresponding multi-channel model and Table 5.10 for the single-channel model on channel Fz. Without aggregation the mean value for cross-entropy is 0.3747 which is better than cosine activation by 0.0043. Although the basis model is worse, there is no clear trend which model is better when aggregation is regarded.

For the single-channel model on channel Fz, the one with cosine activation is better by 0.0072 without aggregation. However, with aggregation and random sampling, only *dvote* and *prod* yield better outcomes. For the sliding

		423N Fz		*453*	
sampling	agg.	median	mean	median	mean
like training	None	0.4082	0.4043	0.3710	0.3704
random	dvote	0.4166	0.4300	0.3888	0.3938
random	psum	0.4444	0.4316	0.3888	0.3922
random	qsum	0.4444	0.4311	0.3888	0.4005
random	pmax	0.3888	0.3911	0.3888	0.4150
random	avote1	0.4444	0.4344	0.3888	0.3938
random	prod	0.3888	0.4166	0.3888	0.3900
s. window	dvote	0.4444	0.4477	0.3888	0.3944
s. window	psum	0.4444	0.4494	0.3888	0.3888
s. window	qsum	0.4444	0.4444	0.3888	0.3944
s. window	pmax	0.3888	0.3900	0.3888	0.4216
s. window	avote1	0.4444	0.4494	0.3888	0.3955
s. window	prod	0.4444	0.4338	0.3888	0.3838

TABLE 5.12: Predictions of models *423N* on channel Fz, and *453* for both sampling schemes and the basic aggregation functions when experiments are performed with cosine loss instead of cross-entropy loss.

window, there is the expected effect of better results. The mean values are on average 0.0156 higher for cosine activation.

> **Answer** (**Q2**, p. 44): For the smoker data set, the cosine loss seems to be better than the classical cross-entropy. Although the basis model was worse in the multi-channel case, after aggregation there was no clear better model. For the single-channel model on channel Fz the results of the basis model as well as those obtained from aggregation with sliding window sampling were clearly better for cosine loss.

5.4.3 Patient-Wise Evaluation

For a given snippet of data, the model estimates the probability of it belonging to each of the three classes. In the standard evaluation, I do not analyze these probabilities directly, but predict the class with the highest probability and analyze the predictions. Figure 5.9 uses the probabilities directly and shows their average for each measurement and each class. Each bar represents one measurement. They are grouped, craving *c* on the left, followed by non-craving *nc* and non-smoker *ns*, as indicated by the prefix. Within each group, subjects are ordered by average prediction quality. For example, the leftmost bar shows the best-predictable measurement from the craving class *c_eb80* with more than 90% correct prediction rate. When the same person *eb80* is non-craving it is also predicted to be craving with more than 90%, as indicated by the rightmost bar of the non-craving block. I assume that the model is able to identify the person and uses this information to predict craving. This is the only subject for which I can conjecture as to why it may behave differently: it is the only left-handed subject in the dataset. Thus, it seems like the handedness might affect the model.

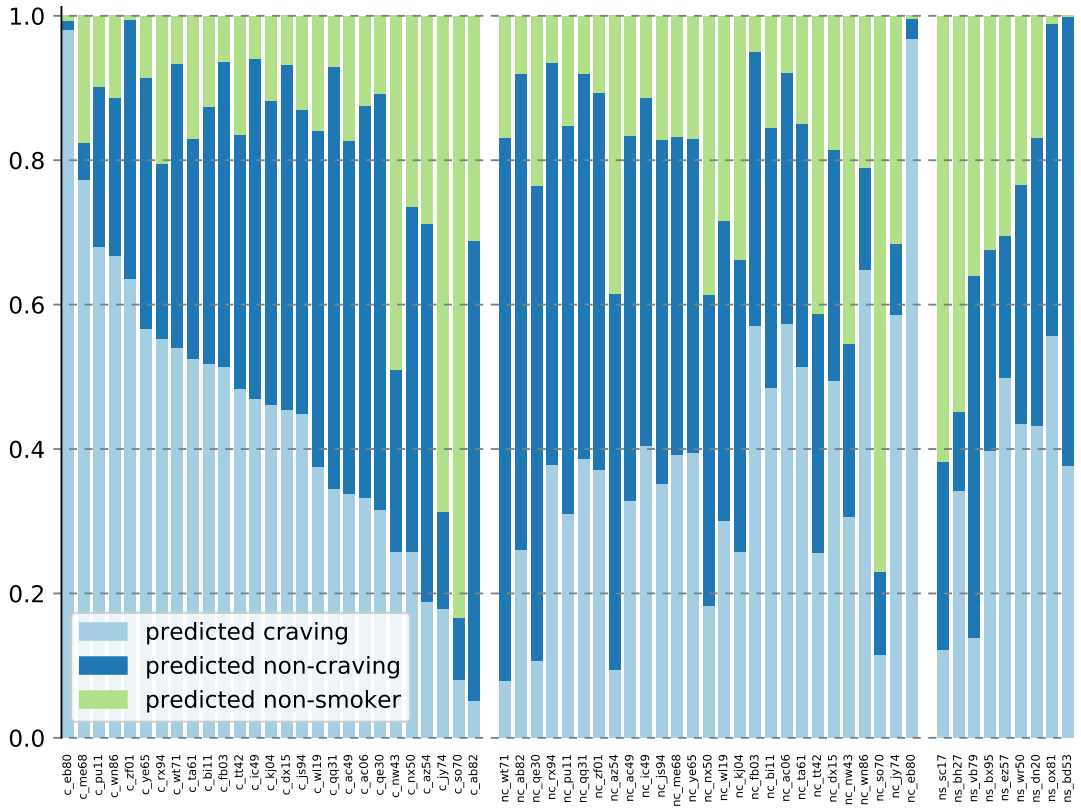


FIGURE 5.9: Bar chart showing the average predicted probabilities for each measurement and all three classes for model *423N* on channel Fz. The actual class is given by the measurement's prefix. The left block shows craving, the middle block non-craving and the right block non-smokers. Each block is ordered with regard to the true positive rate, best predictable on the left.

The overall best predictable smoker *pu11* is the third when craving with 67% correct prediction probability and with 55% the fifth when non-craving. The worst predictable smokers are *so70* and *jy74*. The first is predicted to be a non-smoker with about 80% chance, independent of the craving state. The second is predicted to be non-smoker when actually craving and predicted craving when actually non-craving. The smoker *nw43* seems to be recognized by the model as well as she shows similar prediction values when craving and non-craving — approximately 50% for non-smoker, 25% non-craving and 25% craving. Among the non-smokers, only *sc17* and *bh27* are correctly predicted in most of the cases. Worst-predictable are *dn20*, *ox81*, and *bd53* only predicted to be non-smokers with less than 20%. The models of Pätz [112] were also especially bad for *bd53* and *dn20*. To my model *ac06* looks non-craving when craving and vice versa. One might guess that the measurements were incorrectly labeled, but the labels are correct. Thus I consider this person to be an outlier.

Overall, the predictions show a high variance between subjects and measurements. This could be caused by overfitting. As described in Section 5.4.2, I tolerate overfitting with respect to the training set as long as the performance on the test set is increasing. I assume the model considers many patterns

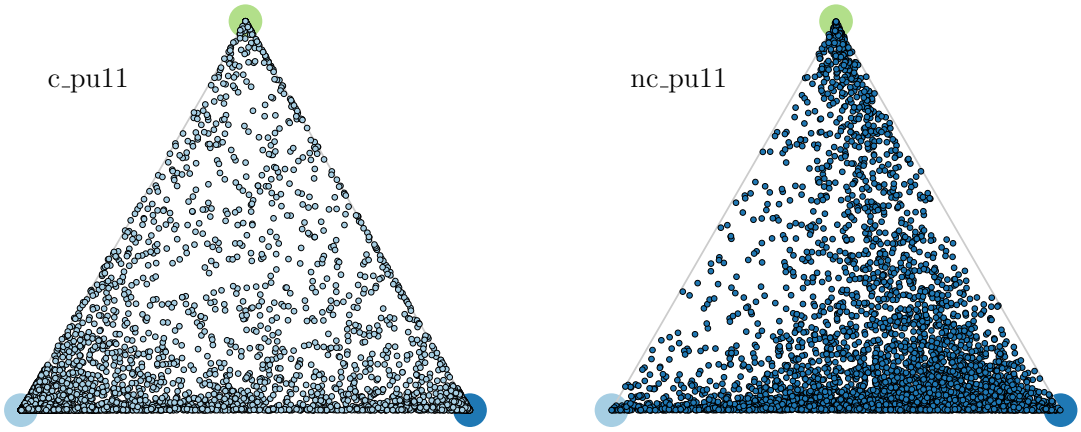


FIGURE 5.10: Point triangle plots for *pu11*

that are person specific or noise related and only few that are related to being a smoker or craving. However, this hypothesis can not be clarified with the current data. To cope with this problem, I suggest the acquisition and analysis of more measurements. This would also open the opportunity to investigate whether the handedness plays a role for prediction models.

Evaluation on Prediction-Level

Question **Q7** (p. 67) asks for the distribution of the snippet quality. It can only be answered directly by simulating the data, as in Chapter 4. With measured data, as given here, the only possibility is to answer the question indirectly, by regarding the individual predictions. This is not direct, because the predictions are also influenced by the chosen model and conclusions can only be drawn on the result of the combination of data and model.

Until now, I have shown only average probabilities of the predictions to give a first, rough overview. This made clear that the differences between measurements may be big, but it allows only few statements on the snippet's distribution. Thus, in the following I present triangle plots of chosen measurements. I also use the opportunity to answer **Q9** (p. 86), which asks for a possible bias towards the triangle's corners that might be caused by the softmax layer.

Figure 5.10 and Figure 5.11 give exemplary triangle plots of the smokers *pu11* and *eb80*. Each snippet is represented by a small dot. As indicator which session is illustrated, the dot's color is chosen accordingly, as before: light blue for craving, dark blue for non-craving and green for non-smoker. The bigger dots in the triangle's corners illustrate the points of 100% prediction for each class.

These pictures give an overview of the actual distribution of the snippets. For white areas it is clear that no snippet was predicted there. For example for subject *eb80*, there are only very few samples on the opposite edge of the real class. In cases like this, *prod* should be a very good aggregation as both other

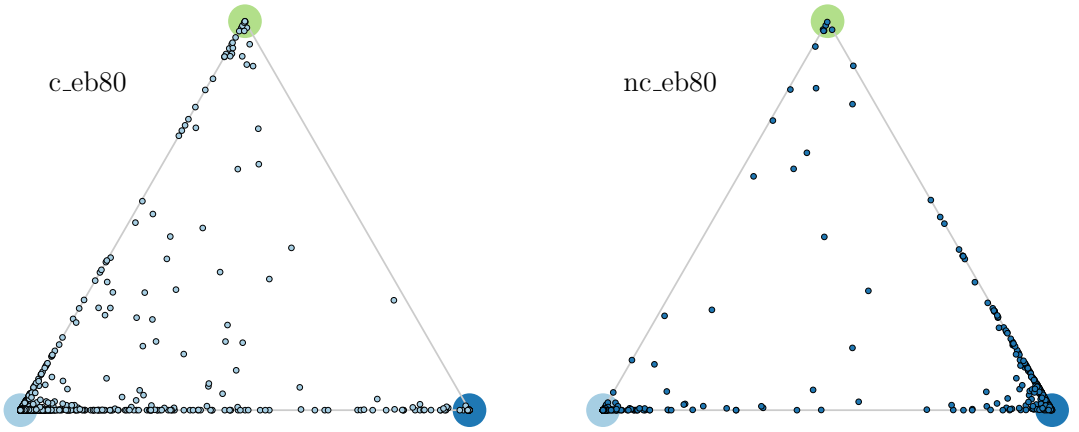


FIGURE 5.11: Point triangle plots for *eb80*

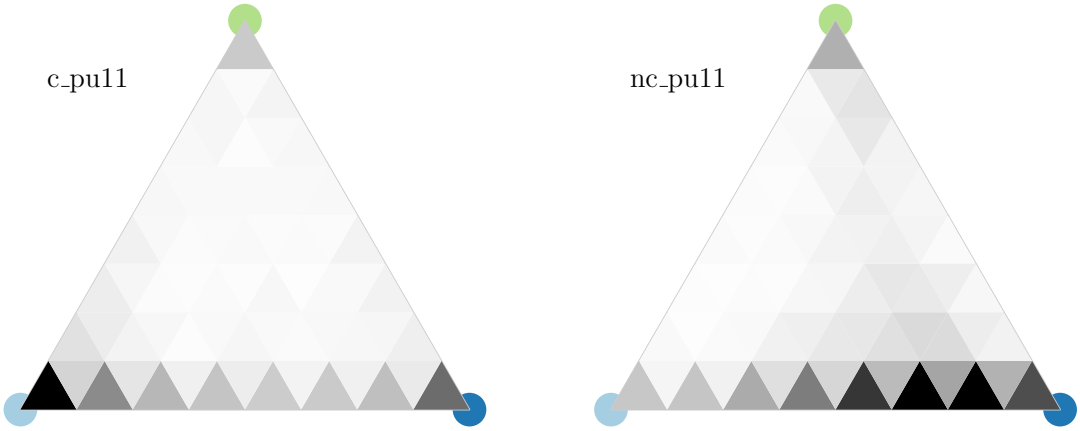


FIGURE 5.12: Density triangle point plots for *pu11*

classes should show values very close to zero. However, in these illustrations, points can lie so close, and even on top of each other that they build a colored area. In these cases it cannot be seen how many points there are. For example it is hard to distinguish which of the non-craving sessions is better on average. Remember that *nc_eb80* has an average value of 90% for craving and *nc_pu11* is the fifth best predictable non-craving session. A second problem was described earlier in Section 2.3.2 and comes with the used random shuffle split cross-validation. Stratification is performed on a class basis, but not on a subject basis. Thus, caused by chance, some measurements are chosen more frequently than others and so there is a higher total number of dots in their triangles.

To cope with these two issues, I implemented a visualization which shows densities in the triangles by coloring areas. Darker areas mean more points, lighter show less points, and colors are chosen with regard to the total number of points. Examples are given in Figure 5.12 and Figure 5.13.

Although the individual predictions are not shown in these figures, the density visualization makes clear that *nc_eb80* is mostly incorrectly predicted to be craving and *nc_pu11* is correctly predicted to be non-craving most of the times. Figure 5.12 also shows that many predictions lie close to the lower

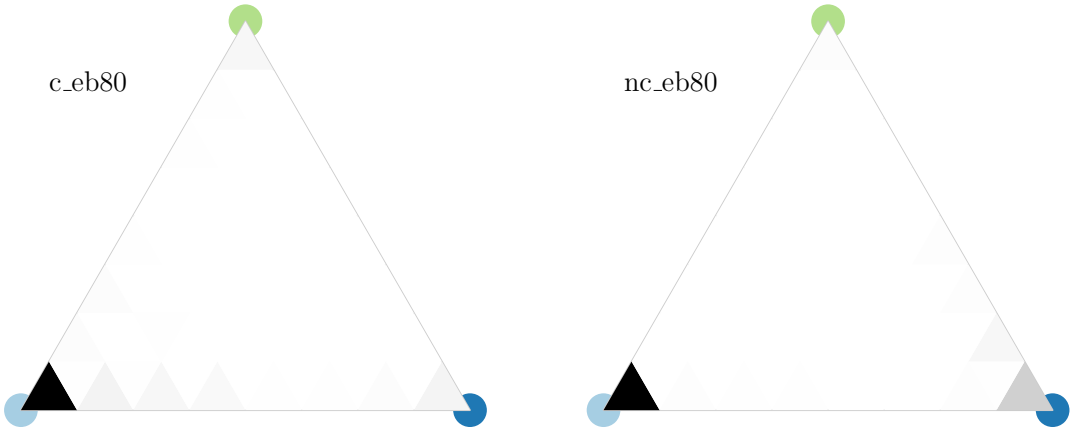


FIGURE 5.13: Density triangle plots for *eb80*

edge, far away from the non-smoker corner. So for this subject, the model excludes the possibility of seeing a non-smoker correctly with high probability. For this model, I show many detailed figures in the Appendix C (p. 185): I start with a violin plot without aggregation function followed by multiple violin plots for sliding window sampling using the different aggregation functions. Then, I provide a bar chart with the averaged prediction for each measurement. Finally, I give details about the prediction's distributions by presenting both types of triangle plots for each measurement. They show that each measurement seems to have their own characteristics. Some have most points close to one or two of the triangle's edges, others are widely spread. But none of them look like the results from the simulation. The source code to create these figures is available in `result_triangles.py`

> **Answer** (**Q7**, p. 67): The snippets' distribution is far from trivial. Some snippets seem better suited than others, some give the wrong classes very high probabilities and thus it seems like they can even mislead the classifier. The distributions depend highly on the subject, there is no clear pattern and none of them look similar to those seen in the simulations.
>
> **Answer** (**Q9**, p. 86): In the performed experiments, there was a bias towards the corners of the simplex (triangle). But in contrast to the simulations of Chapter 4, predictions were often not in the middle of the triangle, but close to its edges. The models seem to have learned criteria to exclude one class instead of criteria that would allow to predict one.

Normally, in machine learning, it is assumed that the data contain features which allow a correct classification. Neural networks are able to find their own features. For a known task, like face recognition, relevant characteristics of a face are known, but engineers allow the network to use many training samples to find their own optimal features. Afterwards, the found features are analyzed and compared to the expected ones. Differences can give very helpful insights: Maybe it gives hints on even *better* features for the task, or it shows possible biases within the training data.

		423N Fz		*453*	
sampling	agg.	median	mean	median	mean
like training	None	0.6972	0.6995	0.7148	0.7021
random	dvote	0.7222	0.7455	0.7777	0.7394
random	psum	0.7222	0.7255	0.7777	0.7322
random	qsum	0.7222	0.7327	0.7777	0.7333
random	pmax	0.6666	0.6927	0.7777	0.7505
random	avote1	0.7222	0.7433	0.7777	0.7400
random	prod	0.7222	0.7077	0.7222	0.7272
s. window	dvote	0.7500	0.7572	0.7777	0.7350
s. window	psum	0.7222	0.7355	0.7777	0.7355
s. window	qsum	0.7222	0.7433	0.7777	0.7350
s. window	pmax	0.7222	0.7011	0.7777	0.7500
s. window	avote1	0.7222	0.7522	0.7777	0.7388
s. window	prod	0.7222	0.7211	0.7222	0.7266

TABLE 5.13: Top2 accuracy overview of models *423N* on channel Fz, and *453* for both sampling schemes and the basic aggregation functions

For an unknown classification task so difficult that the features do not allow for a good prediction, even for a human, it would be an intelligent decision to look for features that allow to exclude one of the classes. Doing so uses the model's full capacity for a possibly simpler task. After correctly excluding one class, the chance for the correct prediction has increased from $\frac{1}{n}$ to $\frac{1}{n-1}$.

Evaluation of Top2 Accuracy

Viewing the triangle plots in Appendix C, the impression can arise that in many cases the model excludes one class correctly. In order to investigate if this effect is only an impression or holds also by the numbers, I consider a different scoring function: Instead of using the class-balanced accuracy, I consider now the class-balanced top2-accuracy. This means that I count a classification as correct if one of the two highest probabilities belongs to the correct class and I respect class balance by triple weights to non-smokers.

Table 5.13 shows the result of this experiment, again for model *423N* for single-channel Fz and model *453* for the multi-channel case. The chances for random guessing are 66.66%. Both models show with 69.95% and 70.21% rather similar mean values when tested *like training*. This is surprising, because the single-channel model was clearly better when considering the accuracy value. The difference becomes clearer when median values of aggregation is considered, which is here clearly better for the multi-channel model. However, sliding window and random sampling show a relatively strong difference for model *423N* — on average sliding window performs one percent better. For model *453* the difference is less than 0.1%.

The aggregation methods show similar trends as before. The summing and voting based methods perform again similarly for model *453*, but the voting-

score function	agg.	mean value reached	random guessing	% of base error rate
top1 accuracy	no	39.71%	33.33%	90%
top2 accuracy	no	70.21%	66.66%	89%
top1 accuracy	yes	43.05%	33.33%	85%
top2 accuracy	yes	75.72%	66.66%	73%

TABLE 5.14: Comparison of top1 and top2 accuracy evaluation

based methods seem to outperform the summation-based methods. Aggregation with *prod* is again worse than these groups. This is opposed to the expected effect of *prod* being especially good. This error is produced because the product is able to get clearer predictions in the considered cases, but the other aggregation functions are also able to predict the correct classes, just with lower certainty. However, it seems like the cases where *prod* is mislead by few samples with very small probability cause an overall negative effect. The *pmax* aggregation is again the outlier. It performs a bit better than the others for the multi-channel model and clearly worse than all functions in the single-channel case.

To clarify whether top2 accuracy or the standard accuracy (top1) show better results, the values created by random guessing need to be considered. In Table 5.14 I summarize again the mean values of the best models without and with the best aggregation (taken from Tables 5.5 and 5.10) and top1 and top2 accuracy, from Table 5.13. Then I compute the percentage of the base error rate, which is a linear scaling measure between 0 for a perfect model and 1 for random guessing. This value allows a fair comparison, which respects the level random of guessing: base error rate $= \frac{1-\text{reached value}}{1-\text{guessing}}$. It can be applied to an arbitrary scoring function, but I apply it only to top1 class-balanced accuracy and top2 class-balanced accuracy.

Without aggregation, models reach with 90% and 89% very similar base error rates. But with aggregation they improve to 85% for top1 accuracy and 73% for top2 accuracy. This clarifies that the hypothesis is correct: when the models are evaluated with top2 accuracy, they do perform better. I conclude that my models — especially when applied with aggregation — are better at excluding one incorrect class than at finding the correct one.

5.5 Input-Output Visualization

The result that EEG data contain indicators on the state of craving and smoking is a new insight for neuro-science. However, more important is the question of which features of the EEG allow for the ability to make good decisions. Free frameworks to visualize are often limited to visualizing features of 2D convolutional layers as pictures. To my knowledge no such framework exists that is able to easily visualize the weights of dense layers.

FIGURE 5.14: Visualization of input and the corresponding prediction

However, instead of visualizing the learned features, as a first step I created a visualization for neuro-scientists that shows the input signal and the corresponding prediction (Figure 5.14). The problem of this visualization is that an input interval is related to three output values, and I want to visualize the effect of changing input and output over time. The three output values changing over time along the x-axis, are shown in the lower of the three plots. As they correspond to predicted probabilities, these functions sum up to 1 at every time point. I use the same color coding for the classes as before, shown at the bottom left. Inputs are shown in the upper plot. Here, with only one curve, I show a single-channel model. For each point in time, the lower plot shows the three output probabilities corresponding to the input interval of length 1024 starting at that same point in time in the upper plot. For example, the leftmost points of the output curve — here showing probabilities of about 65% for non-craving, 35% for non-smoker, and 0% for craving — correspond to the input interval of length 1024 that starts also at the left in the upper plot. The x-axis of input and output are synchronized to show always the same time interval. To enable the user to look for relations of input and output, this time interval can be scrolled and scaled.

This visualization tool is not just of theoretical interest, but also of a practical one for my colleague Sarah Donohue. She carefully looked through the entire data set in order to find which features might be relevant. Unfortunately, she did not find any visual features. With prediction rates of only 40%, this is not a big surprise, but it was worth a try.

As feedback on an earlier version of this program, she requested to add another visualization of the input showing the frequency spectrum after a Fourier transformation, which is drawn in the middle plot. Note that this transformation is created by using only the leftmost snippet.

Additional details are given on the left: the session and subject id and the corresponding number of runs, as shown earlier in the overview in Table 5.2 (p. 108). So the functions drawn in the bottom plot do not correspond to a single run, but show the mean values from *all* runs available for this session. The source code of the visualization can be found in `ResultViz.py`.

Note that result values in the bottom plot are shown for all time points, which is more output than my normal testing scheme returns. Here, I perform tests for *all* possible snippets of length 1024 and call this procedure *extensive* testing. As this takes much more time and memory, I have only performed this few times, and only for models that have already proven to be effective. With results from many more snippets available, I am interested in how aggregated results are affected when comparing standard testing and extensive testing, see also **Q11** (p. 88).

Future work could aim for several improvements of this visualization. For example the plot with the output curves could be extended and also show the curves variance in lighter colors. Or the frequency curve in the middle could be replaced by a spectrogram with the frequencies in the y-axis. This would straighten the overall view, because the upper, middle and lower part of the visualization could then share the same x-axis.

Results on Extensive Testing

For the simulated data, I found aggregations on more snippets mostly leading to better results, when they are independent. Here, they are dependent for two reasons: First, because they stem from the same session and second because they overlap and thus show identical parts of the data. However, more snippets have a chance of containing additional information, because they may show parts of the data that were not shown before and because they may *connect* earlier shown parts.

Table 5.15 shows the results of the two already known models *453* for multi-channel and *423N* for the single-channel Fz, but here for *extensive* testing. For technical reasons, I trained the same models again, and thus also generate additional results *like training*, which show relatively large fluctuations. These are caused naturally by different network initializations and should be respected as well, when comparing the aggregated results.

Model *453* has a worse basis model *like training* (see Table 5.5) by a mean of -0.057 compared to the one gained by *extensive testing*. Consequently, the benefit from aggregating is also smaller. The mean values differ on average by -0.101. The single-channel model shows with $+0.067$ a positive fluctuation compared to the basis model from Table 5.10. With aggregations, the average increase of the mean value is $+0.208$ with *extensive testing*.

Note that I have measured the variance caused by initialization (on one split) at the beginning of Section 5.4 — it was 0.00119. The corresponding standard deviation is approximately 0.034, which is much smaller as the values seen

		multi-channel *453*		single-channel *423N* Fz	
sampling	agg.	median	mean	median	mean
like training	None	0.3750	0.3690	0.3984	0.4038
random	dvote	0.3333	0.3894	0.4444	0.4450
random	psum	0.3611	0.3911	0.4444	0.4494
random	qsum	0.3333	0.3900	0.4444	0.4433
random	pmax	0.4444	0.4172	0.3888	0.4038
random	avote1	0.3333	0.3888	0.4444	0.4450
random	prod	0.3333	0.3800	0.3888	0.4311
s. window	dvote	0.3333	0.3888	0.4444	0.4450
s. window	psum	0.3611	0.3911	0.4444	0.4500
s. window	qsum	0.3333	0.3905	0.4444	0.4433
s. window	pmax	0.3888	0.4122	0.3888	0.4027
s. window	avote1	0.3333	0.3894	0.4444	0.4450
s. window	prod	0.3333	0.3800	0.3888	0.4333

TABLE 5.15: Result overview for the best models for both sampling schemes — here the results of *extensive testing*.

here. It may be that the variance from one split is not a good estimate for the variance that is created by identical models on the same 50 splits, although they only differ in the initialization.

From these values, I draw two main conclusions: First, there are strong fluctuations in the basis model, which make it difficult to draw clear conclusions. Second, as expected the effect of *extensive* testing is clearly positive, but in this case tiny. I assume a reason for this may be that the models are still only a small amount better than random guessing.

> **Answer** (**Q11**, p. 88): Extensive testing showed results that were on average better than the standard testing procedure. However, for this task, the effect is surprisingly small, compared to the effect caused by different initializations. I assume that due to the dependency of the snippets, more snippets do not provide much *additional* information and thus only improve the predictions a little.
>
> **Answer** (**Q15**, p. 109): For extensive testing, the two sampling methods perform similarly. The variance between the two methods is even smaller than for standard testing.
>
> **Answer** (**Q10**, p. 88): The group of sum-based and vote-based aggregation methods performs similarly again. *Prod* is slightly worse again, but *pmax* performs differently: When the model performs relatively badly without aggregation, as in the multi-channel case, then *pmax* is better than the other functions. Otherwise it performs worse.

		original data		permuted data	
sampling	agg.	median	mean	median	mean
like training	None	0.5716	0.5737	0.5166	0.5096
random	dvote	0.5833	0.6016	0.5000	0.5083
random	psum	0.5833	0.6025	0.5000	0.5116
random	qsum	0.5833	0.6025	0.5000	0.5116
random	pmax	0.5833	0.5841	0.5000	0.5175
random	avote1	0.5833	0.6008	0.5000	0.5100
random	prod	0.5833	0.5916	0.5000	0.5041
s. window	dvote	0.5833	0.5983	0.5000	0.5175
s. window	psum	0.5833	0.5975	0.5000	0.5141
s. window	qsum	0.5833	0.5975	0.5000	0.5141
s. window	pmax	0.6666	0.6175	0.5000	0.5391
s. window	avote1	0.5833	0.5966	0.5000	0.5158
s. window	prod	0.5833	0.5941	0.5000	0.5091

TABLE 5.16: Comparison of results from original and permuted data for model *453* and the craving vs. non-smoker task.

5.6 Permutation Tests

When not knowing the relevant features for a task and results are significant but not easily interpretable there might be a problem. The model could have learned correlations within the data that are not characteristics of the actual task. An anecdote example is given in Section 2.3.1. It is difficult, if not impossible, to find out if the result is caused by another effect in the training data. One approach to ensure that the results differ from a classification with arbitrary class labels is to perform a *permutation test*. There, the class labels are randomly permuted and the resulting data are then trained just as the original. While on the training data one expects values better than random guessing due to overfitting, this is not expected on the test set.

In order to minimize the variance caused by the random permutation and class imbalance, once again, stratification needs to be considered. So for each true class, samples should be given fake class labels with respect to the real occurrences. But this is contrary to the effect of subject dependency that forces me to keep all data from the same subject in the same set. Therefore, for three classes with subject dependency it is impossible to perform permutation tests. Thus, I perform permutation tests on a two class problem: non-smoker vs. craving. As for the original data, I also perform experiments for 100 runs with different splits. For greater reliability of the results, the process of permuting, training and testing can be repeated several times with different permutations. However, I performed only one iteration with 100 runs for multi-channel and one for the single-channel model. The permutation I used is given in `data/Smokersshuffled.csv`.

In Table 5.16, I show the comparison of results for model *453*. The permuted data are with a mean accuracy of 50.96% close to random guessing. Thus, I assume that the deviation from guessing is caused by chance. The increased

		original data		permuted data	
sampling	agg.	median	mean	median	mean
like training	None	0.5666	0.5580	0.4566	0.4640
random	dvote	0.5833	0.5716	0.4166	0.4341
random	psum	0.5833	0.5750	0.4166	0.4333
random	qsum	0.5833	0.5750	0.4166	0.4333
random	pmax	0.5833	0.5641	0.4166	0.4533
random	avote1	0.5833	0.5750	0.4166	0.4358
random	prod	0.5833	0.5733	0.4166	0.4233
s. window	dvote	0.5833	0.5900	0.4166	0.4541
s. window	psum	0.5833	0.5875	0.4166	0.4466
s. window	qsum	0.5833	0.5875	0.4166	0.4466
s. window	pmax	0.5416	0.5633	0.4166	0.4691
s. window	avote1	0.5833	0.5891	0.4166	0.4533
s. window	prod	0.5833	0.5858	0.4166	0.4350

TABLE 5.17: Comparison of results from original and permuted data for model *423N* and the craving vs. non-smoker task.

skewness of the distribution can be seen for the aggregation functions, which mostly show higher means. Obviously, 50.96% does not suffice to reach the next higher median value above 50% with aggregation. This would be $\frac{28}{54} \approx 51.85\%$, instead of the achieved $\frac{27}{54} = 50\%$.

The results of the single-channel model *423N* on channel Fz are given in Table 5.17. When evaluated with only one batch, this model reaches only a mean value of 46.40%. Although both models use identical splits, there is a big difference between models using data from all channels, or from only one. With aggregation, the model shows the expected increased skewness of the distribution with smaller median and mean values. Here, all aggregation functions show a median of 41.66%.

In contrast to the models that were trained on the original data, both models show significantly lower results when the data was permuted. So although the features to distinguish craving and non-craving smokers from non-smokers are yet unknown, these permutation tests suggest that the positive results are not a product of chance, but caused by characteristic differences in people's brains.

Chapter 6

TUH Artifact Data Set

The Temple University Hospital in Pennsylvania is one of the few institutions that measures EEG data and makes it available to the public for free. According to them, their goal is to enable deep learning research in neuroscience by releasing the largest publicly available unencumbered database of EEG recordings [57]. Since 2002 they included over 30,000 EEGs that may be used for both research and commercialization [25].

Note that all data are taken from patients of the hospital and that each patient is in the hospital because he/she has a disease. The hospital does not take additional measurements from healthy controls, but compares patients that may have different or multiple diseases. Therefore, the results created by these data might not hold for healthy people.

Data is taken from patients in the hospital and provided for download. The measurements are structured into different data sets that may be overlapping. Some contain epileptic or non-epileptic seizures. Patients may have been treated differently, some take medications others do not. As comparison to the smoking data set, I looked for analogous data sets: EEG data from humans, for which a classification is performed. Unfortunately, I did not find data on addiction and neither a data set for which the classes were constant over time such that aggregation functions could be applied. To be comparable with earlier work, I looked for a data set, for which earlier results from other researchers are available. I chose a task, also relevant for the smokers, namely filtering noise. But before noise can be filtered, it needs to be detected in the signal. Thus this data set is created to build a basis of training samples for machine learning algorithms to learn to distinguish different kinds of artifacts within an EEG signal. Some notes on difficulties with publicly available data sets can be found in Section 7.1.

In Section 2.2 (preprocessing), I describe in detail that EEG signals contain not only information stemming from the brain, but also noise from different sources. Current research with focus on removing noise automatically looks for reliable methods to detect parts of the EEG where noise masks the brain signal. After measurements are taken, experts examine them as multivariate time series and annotate noisy parts of the signal. Normally, this information is used to remove these noisy parts and the cleaned data set is used for the *actual* task. Here, there is no *actual* task as this data set's purpose is to auto-

Label	*eyem*	*chew*	*shiv*	*elpp*	*musc*	*null*	total
patients	140	22	14	140	74	213	213
sessions	166	23	14	97	74	259	259
files	177	28	15	105	99	310	310

TABLE 6.1: Descriptive statistics overview for the TUH Artifact data (Source: [26])

mate the process of finding noisy parts in the signal. Thus, models which use the expert's labels are trained to detect artifacts automatically.

This chapter is structured as follows: I first give introductory information on the data and then describe two approaches from the literature published by Roy [122] and Kim [79]. I analyze their approaches and then describe my attempt to reproduce Kim's experiments and revise his work to improve the found weaknesses. Finally, I extend Kim's work to answer my own research questions.

6.1 Data Description

The data is labeled with one of six labels: The label *null* indicates normal noiseless signal, and the following five labels imply different kinds of noise:

1. eye movement (*eyem*)
2. chewing (*chew*)
3. shivering (*shiv*)
4. electrode pop, electrode static and lead artifacts (*elpp*)
5. muscle artifacts (*musc*)

Examples of how noisy EEG signal looks are given in Section 2.2; These examples are taken from the data set described here. Note that what the data set defines as the class *elpp* contains actually one of three classes, which are all related to electrodes and recording equipment, while the other classes are related to human activity. I assume that these occur infrequently such that the task becomes easier by using this definition instead of using three separate classes. However, even if the classes look similar for humans, this might not be true for the algorithm, which would make the *elpp* class hard to predict.

Each time when data is taken from a patient this is called a *session*. A session can be split into several segments. According to the data set description [26] it is common clinical practice to delete uninteresting parts of the signal. Note that *uninteresting* is a subjective term and I assume that only the parts are removed that show strong enough artifacts to occlude the brain signal. Note that the purpose of this data set is to *detect* noise in *real* data. For future research it might be helpful to create and add new, artificial data samples that show no brain signal but only noise.

Channel	eyem	chew	shiv	elpp	musc	null	bckg
Fp1−F7	6833.80	2671.19	995.80	168.34	2263.65	325826.48	7553.75
F7 −T7	419.60	2717.93	1072.69	280.36	3055.57	329984.93	8781.92
T7 −P7	6.99	2717.93	1101.98	298.68	3239.29	328904.10	10044.02
P7 −O1	11.12	2260.05	798.94	242.60	1317.98	331930.05	9752.26
Fp2−F8	6862.30	2689.56	922.74	212.49	2218.83	325332.50	8074.56
F8 −T8	432.98	2728.75	917.47	550.28	2734.15	328421.34	10528.04
T8 −P8	0.00	2726.44	1008.82	519.49	2716.65	327695.80	11645.79
P8 −O2	0.00	2599.00	926.72	299.87	1841.88	330197.22	10448.30
A1 −T7	0.00	2715.38	968.41	215.98	2782.14	330536.13	9094.95
T7 −C3	0.00	2740.68	1115.36	679.90	2814.34	328623.57	10339.16
C3 −CZ	0.00	2728.53	987.88	1047.91	1781.41	329576.36	10190.91
CZ −C4	6.99	2632.76	922.75	915.24	1814.92	329208.76	10811.58
C4 −T8	6.99	2739.71	1098.70	791.27	2467.57	327495.76	11713.00
T8 −A2	23.53	2596.27	904.49	435.65	2179.67	330299.23	9874.17
Fp1−F3	6849.35	2616.75	889.77	326.59	2161.08	325455.38	8014.08
F3 −C3	337.67	2526.77	964.64	776.45	2201.62	330574.26	8931.57
C3 −P3	33.20	2469.79	984.87	1028.99	1638.87	330712.13	9445.15
P3 −O1	32.82	2467.37	884.04	606.78	965.91	330806.46	10549.62
Fp2−F4	6877.35	2464.29	881.08	225.87	1758.95	325914.26	8191.19
F4 −C4	331.54	2434.58	902.48	654.66	2048.53	330572.58	9368.63
C4 −P4	0.00	2429.50	891.65	1155.75	1556.22	330651.81	9628.07
P4 −O2	3.32	2292.88	833.13	783.59	1067.61	331168.49	10163.98
all	6929.58	2741.19	1245.15	2527.64	4799.00	328070.45	0.00

TABLE 6.2: Overview time in seconds per class and channel

In total, the data stem from 213 patients. From some of them several sessions were taken, and when parts were removed, several segments were created, and one file for each segment. An overview is given in Table 6.1.

Labels are given as time-synchronous event (TSE) files that offer term-based annotations. The format allows to specify a probability for a class in an interval. However, in the given data this value is always 1.0000 and can thus be ignored. Each file contains a list of disjoint time intervals covering the entire session, and holds the respective class label for each interval. For each session, TSE files are given for each channel. There is no time point in the data, for which the channels indicate two different kinds of noise — as *null* indicates clean signal, it is allowed that for a time point, some channels are labeled with *null*, and others with a noise label like chewing. But there is no point, which is labeled to show for example eye movement in one channel and chewing in another channel. Thus, when summarizing the intervals for the channels, one can create a new list of time intervals with annotations indicating the noise to be seen in all channels for each time point. This summary is given in another TSE-file. As this format can be easily read and processed, I created a summary in Table 6.2 that shows how many seconds of noise are given for each individual channel and for the summarizing file. Note that the channels are measured as the signal difference of two electrodes, which is why the column “Channel” contains two electrode positions.

Table 6.2 indicates an inconsistency for the labels. The files that are given for each channel contain a sixth class label *bckg*, which holds "background events", as explained in the data set description [26]. However, it is neither clarified what *background events* are nor why this label does not show up in the files for all channels. When examining samples of the files, I found that the differentiation between the *null* class and the *bckg* class is only made for the label files for the individual channels and not for the one for the whole session.

Eye movement is present for nearly 7000 seconds within the data and if present, it can be seen for more than 6800 seconds in those channels that use data from the electrodes Fp1 or Fp2. Approximately 400 seconds of eye movement is given for F7, F8, F3, or F4. The other channels are rarely labeled as *eyem*, if at all. Thus the discrepancy between the channels is high. This is similar but less extreme for muscle movement: It is present in any channel for nearly 4800 seconds, but it is visible for example in $P3 - O1$ for less than 1000 seconds. The opposite is the case for chewing: if present, can be seen in many channels.

When assuming the correctness of the given labels, these information on homogeneity of the channels is important for the experiments performed on single channels in Section 6.4. There is a dilemma: In order to create results comparable to the models that use data from multiple channels, I need to use the same labels (seen in the *all* row). On the other hand, I should use the labels for the given channels, as otherwise it is possible that noise is present at this time point, but is not visible in the given data. Consequently, the model has no chance to train properly and thus performs worse than the multi-channel version. I found no good solution for the dilemma. But as I focus on comparability of the models, I must use the same labels. Therefore, this effect should be considered, when regarding the results of the single-channel models.

6.2 Earlier Results on this Data Set

Roy's experiments

Two publications work with this data set and generate baseline results that allow a comparison to my own results later on. Roy [122] has 'set the benchmark' in his short paper. Unfortunately, his results are not reproducible, as many of his processing steps are neither motivated nor explained. I add notes on reproducibility during the description of his processing.

He used a patient level split into training (60%), validation (20%) and test set (20%), but does not give an overview of which patient is in which data subset. Additionally, the description of the result table indicates that he used a cross-validation and averaged over 5 runs, but sadly omits the details.

He transformed the given data to the transverse central parietal (TCP) montage system (described in Section 2.1 and depicted in Figure 6.9, p. 163). Then he split the data into one second snippets with a 0.75s overlap across all channels. Note that it is only possible to create these overlapping snippets, if both contributing seconds have the same label. For these snippets Roy computed the Fast Fourier Transformation. The transformed data is clipped to remove the signal outside the interval from 1 to 24 Hz. After a normalization across frequency buckets, for which neither their number nor their sizes are given, he creates the matrix consisting of the correlation coefficients between all channels. From the upper right triangle of this matrix — supposedly because it is symmetrical —, he computes the absolute values of the eigenvalues and uses them as his features. The eigenvalues correspond to the relative variances in the PCA space, but their direction may differ considerably. Thus, it is not intuitive to me why one would use the eigenvalues but ignore their directions. However, they are used as input to the following eight supervised learning algorithms: AdaBoost, Gaussian Naive Bayes, k-Nearest Neighbor, Linear Discriminant Analysis, Multi-layer Perceptrons, Random Forest, linear classifier with stochastic gradient training (SGD classifier) and XGBoost.

To find the optimal set of hyper-parameters, he used the HyperOpt meta-algorithm [10]. He does not describe how he applies this software, but he only references the work of Bergstra et al [10]. Hyperopt is a python library that uses Bayesian optimization to find optimal model hyperparameters. Because of the class imbalance of the data set, Roy uses under-sampling on the most frequent class, but again skips the details.

Although the results given by Roy [122] are not reproducible, they do serve as a benchmark. He lists sensitivity rates for each class and algorithm as well as values for weighted-F1 and accuracy for each algorithm. His best algorithm is LDA with an accuracy of 71.43%. Worst performance was by AdaBoost, with 62.57%. The shivering class is only given in 4,005 of his snippets and is thus strongly underrepresented. Compared to more than 1.3 million snippets of the majority class, this is a factor of more than 300[1]. Therefore it is not surprising that most of the algorithms show specificity of about 3% for *shiv*.

Kim's Experiments

The work of Kim [79] is reproducible. Especially helpful is the fact that his source code is publicly available which makes it a valuable basis for my own experiments. The following steps are explained for his work, but I use similar ones for my own experiments later on. Differences between Kim's and my work are elaborated in the next subsection.

The EEG measurements are taken with three different reference electrodes: *averaged reference* (AR) uses 22 channels and their average as reference, AR_A,

[1] The strong difference compared to the data used by Kim (Table 6.3) is caused by the used overlapping: No overlapping by Kim and 0.75 seconds of overlapping by Roy.

Label	*eyem*	*chew*	*shiv*	*elpp*	*musc*	*null*	total
Occurrences	7,471	2,727	1,338	2,663	4,892	327,222	344,975
Percentage (%)	2.17	0.79	0.39	0.77	1.42	94.85	100

TABLE 6.3: Label statistics of the original data (see also [79, p. 31])

is similar to AR, but uses only 20 channels, and *linked ear reference* (LE) uses 22 channels and the mean of the two channels at the ears as reference.

Kim [79] loads the average reference as montage system and just as Roy [122], converts the signals to the *transverse central parietal* (TCP) montage system, as recommended by the temple university (the creators of the data set). Files taken with the linked ears reference can easily be converted to the AR system, but files using the AR_A system are missing the two channels from the ears (A1 and A2) for the conversion. As this occurs only for 7 of 310 files, Kim discards them. This explains the different number of total files used by Roy (Table 6.1) and by Kim (Table 6.5).

Roy uses the data in frequency representation created by a Fourier transformation. Kim on the other hand uses the data directly in their representation as time series. Thus all measurements need to be passed to the neural network not only in the same length, but also in the same sampling frequency. The given measurements show varying frequencies: 250 Hz, 256 Hz, 480 Hz and 500 Hz, for which the first is used the most often. Thus, Kim chooses 250 Hz as fixed sampling rate — presumably to minimize the effort needed for resampling. Measurements with different rates than 250 Hz are resampled by a Fourier transformation (see Section 2.2.2). Kim creates snippets of one second length for each full second of given data without overlaps, and each snippet consists of data from 250 time steps. The ending of a session that does not build a full second as well as those snippets that lie at a transition of two classes are not used.

Table 6.3 shows how many snippets belong to every class, when snippets of one second are used. From a total of 344,975 snippets, 94.85% are labeled as clean signal (class *null*), while each other class adds less than 3% of the snippets. This highlights the strong class imbalance in this data set.

While Roy only states that he used under-sampling of the majority class without replacement and omits details, Kim describes in detail how he uses subsampling to cope with the class imbalance: For each file, he includes only every 30^{th} snippet of class *null* in his data set, reducing the occurrences to 10,763. Table 6.4 shows the number of occurrences and percentages with respect to the new total of 29,854 snippets. The shivering class is so infrequent that none of Kim's models predicted shivering and thus he left this class out in the final evaluation.

Kim's publicly available source code was very useful in order to reproduce his results. Unfortunately, the source code only works on windows as operating system. Thus I had to replace some functions to make it work under Linux, which is the operating system on the server I use.

Label	*eyem*	*chew*	*shiv*	*elpp*	*musc*	*null*	total
Occurrences	7,471	2,727	1,338	2,663	4,892	10,763	29,854
Percentage (%)	25.03	9.13	4.48	8.92	16.39	36.05	100

TABLE 6.4: Label statistics of the subsampled data (see also [79, p. 36])

	training	validation	test	total
Kim's split	224	23	56	303
My split	222	27	54	303

TABLE 6.5: Number of files in Kim's and my own split

Further, Kim's splitting procedure has two drawbacks: The problem that I explained in Section 2.3.2 occurs: Each subject should be treated as independent group during the split. Further, the different kinds of noise are not evenly distributed between the measurements. Therefore, finding an optimal split, that considers groups and stratification, is NP-hard and in contrast to the data set of Chapter 5, the given instance does not allow a fast solution.

Kim's approach is to ignore the stratification criterion. He randomly takes patients, for which approximately 75% are used for training, 10% for validation, and 15% for testing.

This leads to a second drawback I could not fix: Kim's split into training, validation and test set is not reproducible. Unfortunately, the source code does not set the seed of the random number generator for the package *random*, which is used for generating this split. It does, however, set the seed for the *numpy* package. As a consequence, whenever the code is run, it creates a *different* split.

In order to stay as close to Kim's solution as possible, and to avoid solving the NP-hard split problem, I ran his split function several times in order to get a similar distribution as Kim. The exact numbers I used in my experiments are given in Table 6.5.

In Kim's work, there is a small inconsistency: He describes that he normalizes the data, before relaying to the neural network. Therefore, he calculates the mean and the standard deviation of his training set and reports them in his text to be -1.5977595 and 219.39517. In his source code, these values are not computed at each training run, but hard coded to be -0.869822224 and 196.00111. I assume that one set of values refers to the subsampled data, while the other one was computed on the original data with all snippets from the *null* class. So before training, he normalizes his training data to have a mean of 0 and a standard deviation of 1. Note that these normalization values are also used for validation and testing although the actual values might differ. This makes sense, as in a real world scenario the values from validation set and test set are unknown.

Kim's evaluation shows some weaknesses: First, he only performs one split and no cross-validation. Second, he ignores the effect of random weight ini-

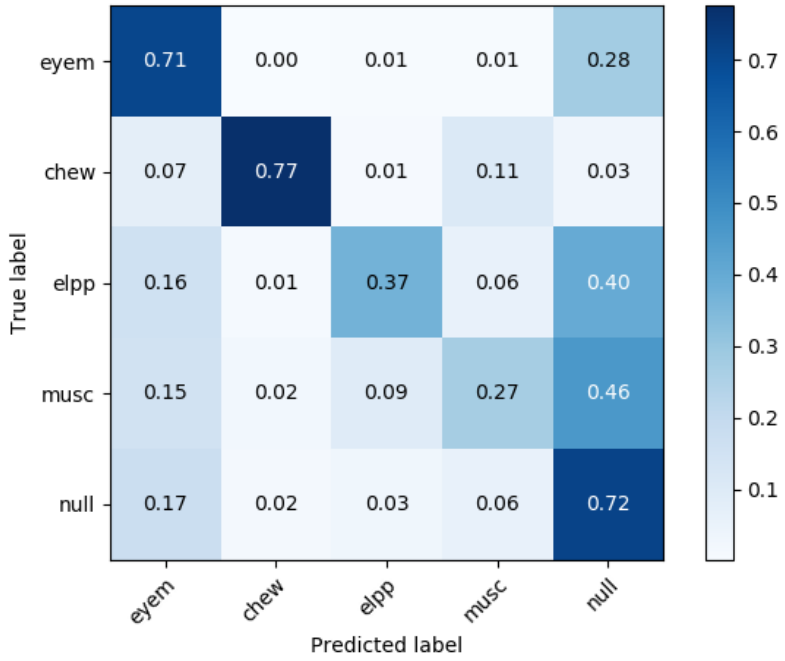


FIGURE 6.1: Confusion Matrix of Kim's Deep CNN Model [79, p. 51]

tialization and shows only the results of one single run. So it is impossible to know if this result is of an average run or if he chose to report the best result of several runs.

He evaluated several models and finds that from several recurrent networks, SimpleRNN, Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) the latter performs best. His best recurrent model uses 50 LSTM units and achieves an accuracy of 58.01%. Networks with 100, 200 and 250 LSTM units were also tested but they overfit and show worse results on the test set. In the following, I give more details on the convolutional model he calls *Deep CNN*.

He reports the accuracies to be 94.72% for training, 44.30% for validation and 65.17% for the test set. His confusion matrix of the test data is depicted in Figure 6.1. It is common to have very high accuracies on the training set, but the results for validation and test set should only show minimal variation, as they are both independent sets and should stem from the same distribution. Another reason for stronger fluctuations might be the small number of patients in the validation and test data set (Table 6.5). Usually, there is a third possibility for big difference between validation and test set, namely overfitted hyper-parameters. But this holds only if the performance on the validation set is higher than on the test set, not for the reverse case seen here. This curiosity indicates a problem in the source code. As my own experiments use parts of this code as well, I investigate this problem on my own split in the following subsection.

In summary, the results by Roy [122] are with 71% accuracy significantly better than the ones produced by Kim [79]. Thus it can be assumed that Roy's

data set	mean	std.
training	-0.953436	214.958
validation	-0.034939	132.796
test	0.608496	271.381

TABLE 6.6: Means and standard deviations of my split

preprocessing is superior: overlapping snippets creating more training samples and using features built from the Fourier transformation instead of the untransformed time series.

Revising Kim's Experiment

To reproduce Kim's experiments, I downloaded the original files from the Temple University Hospital and applied Kim's source code. Minor changes were necessary to cope with Linux as the operating system. As the first step, the original files are read in and those that do not match 250 Hz for the sampling rate are resampled to 250 Hz using a Fourier transformation. As described in Section 2.2, the run-time of this transformation can be very long, especially when the number of time points in the measurement has only few prime factors. This showed up in practice when I ran the code, as it processes the files iteratively one by one: The first few files were processed within seconds and then the program seemed to be stuck, running several minutes, until I canceled the process. Finally, I used all 24 available cores to process the files in parallel, which then took only few minutes in total.

Kim's results seem to be generated by only a single run, thus it is questionable if they suffice as reliable base line. However, with his source code available, I used a single split with identical parameters and ran his experiment again, repeating it 50 times with different, random initializations for the weights. Believing that Kim reported results from one run, I expect that my results are distributed around 65% accuracy, the value reported by Kim.

The actual results differ: 37 of the 50 repetitions show an approximate Gaussian distribution, but the mean value is smaller — about 61%; 13 models have converged predicting always the majority class, resulting in approximately 31.55% accuracy on my test set. By checking the logs of the training, I verified that these models were stuck in a local minimum, showing no significant changes on the quality during training, neither on the training nor on the validation set. This shows the effect mentioned in the Section on initialization (2.4.8). Those of the 50 models that were not stuck achieved an average accuracy of 64%. So I conclude that 1% difference in accuracy can be explained by the different split and the possibility of him only presenting his best result.

In order to cope with the problem of models being unable to learn, I apply a callback on the final experiments that reduces the learning rate on plateaus (see Section 2.4.13). When four consecutive epochs show reduced validation

loss, the learning rate is reduced by a factor of 5. As preliminary experiments with Dropout (Section 2.4.11) were not successful, I decided to use early stopping. This method combines two positive effects: First, it prevents overfitting by memorizing the best model (with regard to validation loss) during training and use this model in the end, instead of the last one, which is more likely to be overfitted. Second, it speeds up the training process. As preliminary experiments with 100 epochs had shown that the final model was mostly found in the early epochs, I set the callback to 15 epochs. Finally I save the model with the smallest validation loss. I use all of these adaptations in the following experiments.

Possible Reasons for Kim's Curious Validation Result

I hypothesize four possible reasons for the curiously low validation accuracy in Kim's work:

1. One explanation for a diverging result between training and testing is a curiosity in the implementation of the layers for BatchNormalization and Dropout. They perform differently in training and testing mode. For BatchNormalization, there is another known issue in the Keras framework: until version 2.1.3 it kept updating minibatch statistics although the layer was frozen. Luckily, I use Keras version 2.2.4, thus I can exclude this possibility. I explain details on this "feature" in Section 2.4.10.

2. Kim's implementation of the normalization applies the normalization of the data set explicitly during training and testing, but in his source code, the optional parameter that applies the validation set during training is not set. Thus, it is possible that he forgot to normalize the validation set when he applied this test. When I applied this error in my code, the validation accuracy was even lower than 40%.

3. Kim may have picked the one run with the best accuracy on the test set. As his work gives no hints on how he performed hyperparame-

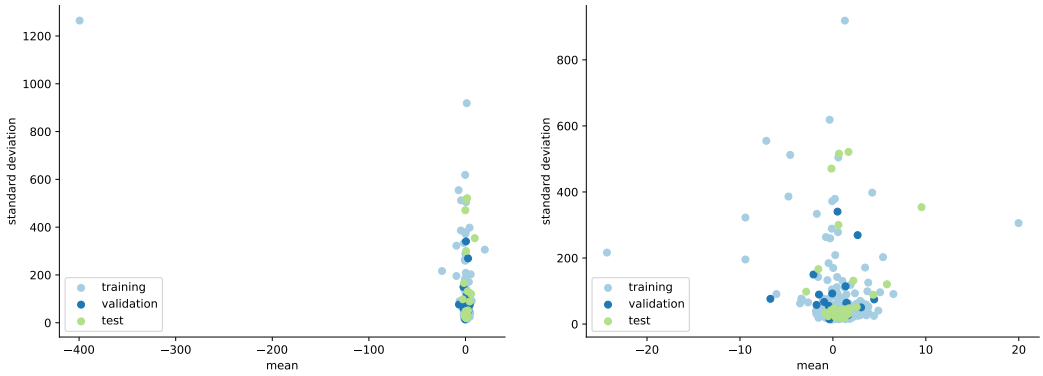


FIGURE 6.2: Mean and standard deviation of samples from the epilepsy data set, 1 point per patient, left with all patients, right with one outlier removed

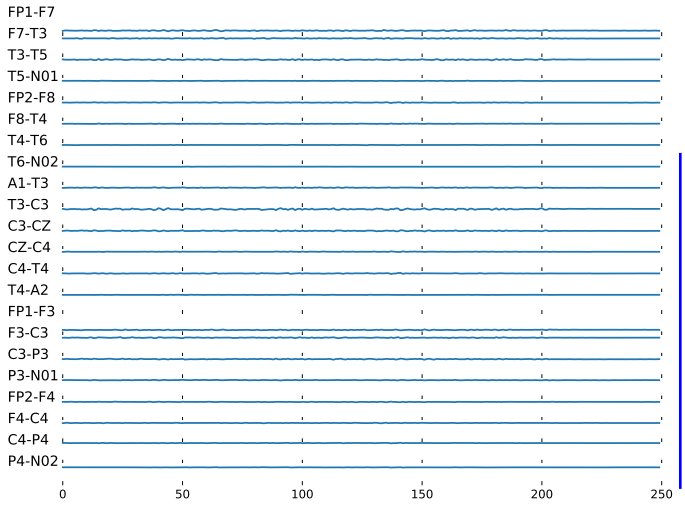


FIGURE 6.3: EEG sample taken from subject 00003849. This sample is this subject's snippet with the largest scaling factor (435) of the y-axis.

ter optimization, it is possible that validation accuracy may have been irrelevant in his choice of the final model.

4. One cause might be an inhomogeneous distribution of the data. As I cannot reproduce Kim's split, I investigate the results created by my own split: The model with highest test accuracy of my 50 runs shows 69.39%, 56.39% and 66.38% training, validation and test accuracy. The relatively small training accuracy can be explained by early stopping — but not the 10% lower accuracy on the validation set. The error might stem from the fact that the factors used to normalize the data are taken from the training set, but the actual values differ significantly. Thus, I compute the actual mean and standard deviation for the training, validation and test set, given in Table 6.6. Although independence between these data sets must be assumed, the standard deviations of validation and test set differ by more than a factor of 2, which is huge. So I look at the variances and standard deviations of each patient. Figure 6.2 shows scatter plots of the entire data set that show the mean and standard deviation of all channels summed up per subject. The left plot shows the whole data set, with one patient in the upper left corner being an extreme outlier with mean of -400 and standard deviation of over $1,200$. In order to make the distribution of the remaining patients clearer, I plotted them again on the right without the outlier. Most points lie at mean values between -3 and 3 and show a standard deviation smaller than 200. Nevertheless, there are still outliers with means greater than 20 or smaller than -20, or standard deviations up to $1,000$; So in both dimensions some patients differ by a factor of 5 or more. This can explain the differences in my split. And assuming a similar split for Kim, this could also explain his low validation accuracy.

Interested in the one extreme outlier, I was able to reconstruct that this patient has the ID 00003849. I visualize his snippet with highest scaling factor,

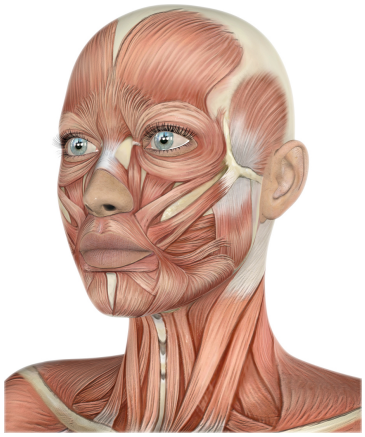


FIGURE 6.4: Head with muscles, Designed by kjpargeter / Freepik, (Source: [70])

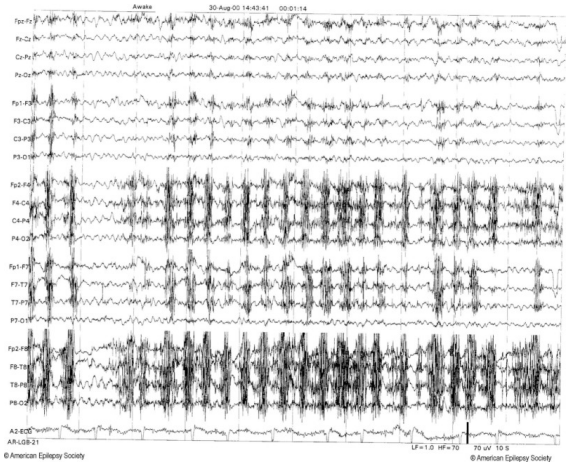


FIGURE 6.5: Chewing artifact visible in many EEG channels (Source: [16])

similar as in Section 2.2 (Figure 6.3). Because of the high scaling, there is hardly any visible brain signal. But the high variance, that marks this subject as outlier is visible in channel Fp1. This channel shows extremely small values. I assume this is caused by a loose electrode or a malfunctioning signal enhancer of this electrode. Kim's normalization is not robust towards errors like this. He should have filtered outliers in advance, when computing mean and standard deviation. One could also consider differentiating between channels; extremely noisy channels could have been interpolated. Thus, a malfunctioning electrode, combined with missing filters and non-robust preprocessing probably induced these implausible results.

6.3 Own Experiments

In the following section, I emphasize the questions I want to answer on this data set and how I approach them. After reproducing Kim's results as a basis, I modify his convolutional networks to answer the following four questions on model variations for this data set:

Q3 (p. 58) Is it sufficient to change the initialization to the identity function instead of using a skip connection?

Q4 (p. 61) Does enhancing convolutional blocks with skip connections improve the performance when performing EEG time series classification?

Q5 (p. 61) Does the network perform better when channel-wise 1D convolutions are added at the beginning?

Q6 (p. 61) Does the network perform better with a skip connection bypassing the channel-wise 1D convolution?

Questions **Q5** and **Q6** were already answered for the smoker data set in Section 5.4.1. In addition to the questions on model structures, I am also inter-

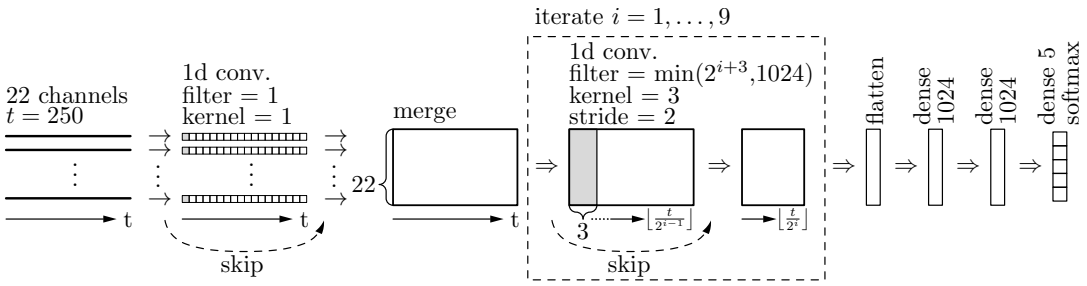


FIGURE 6.6: Network structure for my most complex model

ested in the actual content of the data set. Question **Q8** (p.71) asks for the brain areas best suited for the given task. The task here is to detect artifacts in EEG channels. Eye movements are most common in channels close to the eyes: Fp1, Fp2 and less affected, farther away F7 and F8 (as seen earlier in Table 6.2). The muscles along a human head are depicted in Figure 6.4. It can be seen that chewing causes not only muscle movements inside the jaw, but e.g. also movements close to the temples, as this is where the muscles are attached. Thus, one might assume that channels nearby (T7, T8, P7, P8, F7, F8) are most affected. However, the example from Britton et al. [16] in Figure 6.5 as well as the overview shown in Table 6.2 demonstrate other channels are also affected by chewing, but less severely.

My approach to question **Q8** is to generate models that use data from single channels, only. Further, I visualize the results on a head-plot (described in Section 2.1) to verify whether the assumed channels from above are actually the ones for which noise can be detected best. Finally, I inspect the confusion matrices of single-channel models to check whether the chosen channel relates to the expected source of noise.

Network Structure

The network structure used is visualized in Figure 6.6. It shows my most complex extension to Kim's model. It uses residual blocks, applies channel-wise 1D convolutions and an additional skip connection. A convolutional block consists of a 1D convolution with the given number of filters, followed by BatchNormalization (omitted in the Figure) and MaxPooling with pool size 2. A residual block works similarly, but adds a skip connection, which bypasses the convolutional block. The following list emphasizes on the network's layers, not just for the most complex, but for all variants of the model:

1) Input layer: 22 channels with 250 time steps
2) (optional) channel-wise 1D convolution
   Initialization: default (Glorot Uniform) / custom (to answer **Q3**)
3) (optional) skip connection around channel-wise 1D convolution
4) Merge layer

5) 9x: convolutional / residual block (stride: 1, filter size: 3)
filters: 16, 32, 64, 128, 256, 512, 1024, 1024, 1024
6) Flatten
7) Dense layer: 1024 neurons
8) Dense layer: 1024 neurons
9) Dense layer: 5 neurons (Softmax)

One drawback I kept from Kim's approach is that it loses some information due to rounding down, when applying MaxPooling (see Section 2.4.6). This is caused by a combination of several factors: The model uses snippets of length 250. This value was chosen in order to have one second — the interval for which the labels are given — and the data is sampled at 250 Hz[2]. But the MaxPooling of each convolutional (or residual) block pools two time steps into one. As a negative consequence, information is lost several times because rounding down in multiple blocks is used by default. Information loss due to rounding could have been avoided for example by resampling the data to 256 Hz or by using snippets of 1.024 seconds, or by forcing the model to round up. Rounding up seems to be the easiest solution, as it can be done easily, just by setting one parameter differently for the pooling function. Taking snippets of 1.024 seconds would also be possible. To resample all signals would have the disadvantage that it is additional effort, especially since most sessions are taken in 250 Hz. However, Kim used none of these solutions and I created by models the same way in order to be able to reproduce his results as close as possible.

Hyperparameters

- Optimizer: Adam
- Learning rate conv.: 0.001, res: 0.0001
- Loss function: categorical cross-entropy
- Maximal number of epochs: 100
- Reduce learning rate on plateau:
patience : 4, factor : 0.2, monitor : validation loss
- Early Stopping: after no improvement for 15 iterations
- Batch size: 32
- Total number of parameters (most complex model): 11,548,185

[2]Note that for the smoker data set, I use mainly snippets of approximately two seconds length. There, I use the native sampling rate of 508 Hz, as data is given is this form for all measurements. There I can also easily create overlapping snippets as the labels do not change during measurements.

model	cw1Dc	skip	custom init	# failing models	avg acc. (all models)	avg acc. (good models)
conv	no	—	—	18	51.37%	62.52%
conv	yes	no	no	19	49.17%	59.97%
conv	yes	no	yes	19	49.51%	60.52%
conv	yes	yes	no	13	55.13%	63.40%
res	no	—	—	0	63.81%	63.81%
res	yes	no	no	0	63.22%	63.22%
res	yes	no	yes	0	63.55%	63.55%
res	yes	yes	no	0	63.41%	63.41%

TABLE 6.7: Accuracy predictions of different model for the seizure data set

6.4 Results

Experiments without callbacks (Section 2.4.13) and with residual blocks show an interesting behavior: These networks show accuracies above 70% on the training set as well as the validation set; they learn even faster than the convolutional variant. But on the test set they show an accuracy of only 31.6%, which is the base rate when always predicting the *null* class. 48 of 50 models always predict the majority class on the test data; the remaining two models predict other classes in very few cases. This means that they are strongly overfitted with respect to training and validation data, but are unable to generalize. This effect seems similar to the one described in Section 2.4.9.

For convolutional models, the callbacks worsened the problem of models predicting always the majority class on the test set: Without callbacks 13, with callbacks 18 models kept predicting the majority class. The remaining 32 converged at about 62.52%. For residual models, the callbacks solved the overfitting problem — no models predict only the majority class. So residual models with callbacks give the best solutions. The overview of models and average predictions with applied callbacks is shown in Table 6.7.

> **Answer** (**Q4**, p. 61): Yes. Enhancing convolutional blocks with skip connections does increase the performance of the model: First, the number of failing models is reduced to 0. Second, even when only considering the good models, the accuracy improves. But note that callbacks that reduce the learning rate on plateaus and apply early stopping are necessary to prevent overfitting.
>
> **Answer** (**Q5**, p. 61): Unclear. Channel-wise 1D convolutions show no significant differences when residual networks are used. For convolutional models the results vary; some are better, some worse with channel-wise 1D convolutions.

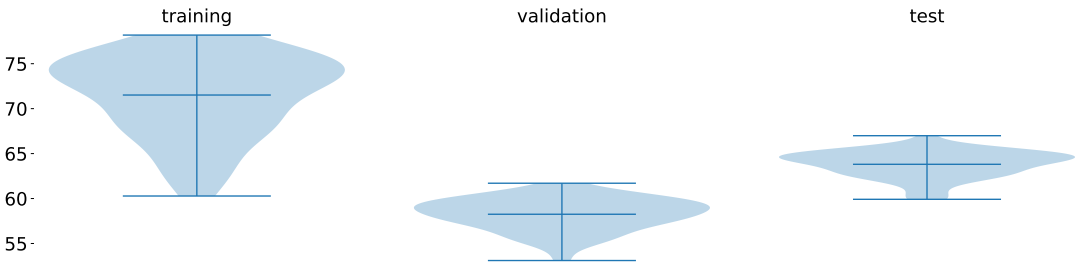


FIGURE 6.7: Violin plots showing accuracies(%) for all 50 runs of the res model without channel-wise convolution for training, validation and test set (left to right).

> **Answer** (**Q6**, p. 61): The network does not perform worse in any case. Table 6.7 shows that only insignificant variances were found for models with no stability problems (here the residual networks). For plain models there is a significant improvement when both channel-wise 1D convolution and skip connection are applied. Then the average accuracy is better by nearly 1% and the number of failing models is reduced from 18 to 13. This effect is particularly curious, because each of the two basis models (row 1 and row 2) perform worse than their combination (row 4). I can only speculate about the reasons of this effect. It is possible that it is caused by chance. Another hypothesis is that the convolutions have a positive effect on the model stability, but that this effect is more than canceled out if it is not combined with a skip connection.
>
> **Answer** (**Q3**, p. 58): No. Using custom initialization instead of a skip connections produces similar or worse results.

Figure 6.7 shows violin plots visualizing the distribution of the model accuracies for the 50 runs of the residual model without channel-wise convolutions for training, validation and test set. The inhomogenous data distribution can be seen here again with 6% lower average accuracy on the validation set than on the test set. All three distributions show a negative skew. On the test set, the mode lies at about 64.5%, the mean value at 63.81%. The worst model reaches 60% accuracy, the best one nearly 67%.

The confusion matrix of the best model is depicted in Figure 6.8 and shows that three classes are detected with high accuracy (over 75%): chewing, muscle artifacts and the *null* class that contains clean signal. When chewing is mispredicted, it can be predicted to be any other class with approximately equal probabilities. The other classes are often mistaken to belong to the majority (*null*) class. For muscle artifacts this effect is smaller, for *elpp* and *eyem* it is bigger. Although eye movement is the second most frequent class, it is detected correctly in only 44% of the cases and it is misclassified to be clean signal in over 40%. *elpp* contains samples showing electrode pop, electrostatic and lead artifacts. To be precise, samples there belong to one of three classes which are combined in *elpp*. After the shivering class is removed, *elpp* is the class with fewest samples. So for each of the three actual classes the number of occurrences is even smaller. Thus, detecting only 13% is not surprising.

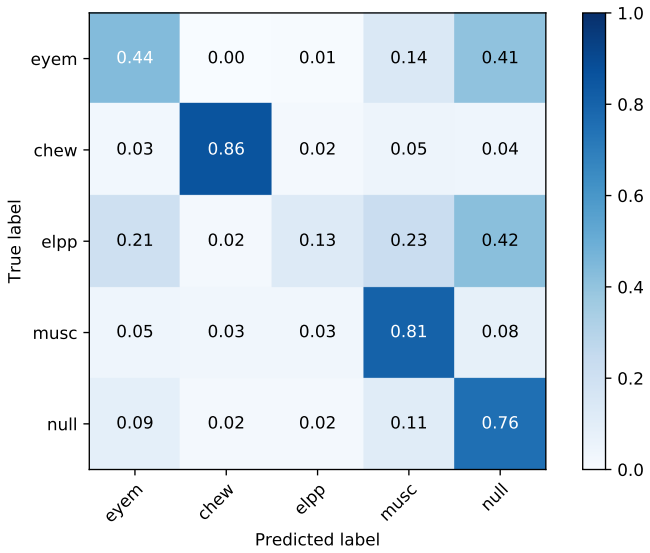


FIGURE 6.8: Confusion matrix of the best model from all 50 runs of the residual model without channel-wise convolution.

loss function	model	# failing models	avg acc. (all models)	avg acc. (good models)
cross-entropy	conv	18	51.37%	62.52%
cross-entropy	res	0	63.81%	63.81%
cosine	conv	42	33.82%	50.84%
cosine	res	9	57.31%	63.06%

TABLE 6.8: Accuracy comparison between models with different loss functions: categorical cross-entropy and cosine loss. All models use no channel-wise 1D convolution

Evaluation of Cosine Loss

As explained in Section 2.4.4, I also want to investigate which of the two loss functions performs better, the cosine loss or the softmax function in combination with the categorical cross-entropy. Note that for the given time series data, I could not apply a network structure with many residual connections, as it is common for pictures. Further, I kept the Adam optimizer and did not apply stochastic gradient descent, as proposed for pictures by Barz et al. [7]. Further, I did not apply a learning rate schedule, but kept the starting learning rates at 0.001 for the convolutional model, and at 0.0001 for the residual model. I also applied the earlier described early stopping and reduce the learning rate on plateaus.

The results for my 50 runs are given in Table 6.8. It shows the results of the convolutional and the residual models and for each of them the typical cross-entropy loss after the softmax layer and the proposed cosine loss. For both models, the number of failing models is increased: for the convolutional model from 18 to 42, for the residual model from 0 to 9. This results in an

decreased average accuracy for all models. When considering only the good models, the average value is worse for the convolutional but similar for the residual model.

> **Answer** (**Q2**, p. 44): For the artifact data set, the cross-entropy loss yields better results than the cosine loss. The number of failing models is higher, and the average accuracy is lower for the cosine loss.

Single-Channel Models

I also want to investigate the performance of single-channel models. For this data set, information is available on when noise is present and in which channel it occurs. Unfortunately I could not find any related work that performed this task on single channels. To stay comparable with the results obtained before, I apply the same expert annotations for models that work on data from all channels. Therefore, the dilemma from Section 6.1 occurs: In some cases, there is noise annotated that, but not visible in the given channel. This is one reason to expect worse results, here. The second reason is that the number of samples is identical, but as the model receives only the input from one of the 22 channels, each sample holds only $\frac{1}{22}$ of the data.

However, my focus of interest is not to create optimal models for this specific problem but in insights on how well predictive models perform when instead of several (here 22) channels, only one is given. For this comparison I need to use the same annotation data, stemming from the TSP file that holds information on *all* channels not just for one specific. I use the same models as explained for multiple channels, with the difference that I use the data from one channel and pass this data to each of the 22 input channels of the model. Thus, single-channel models have the same size as multi-channel models.

Table 6.9 shows the results. When montages always apply the same reference electrode, this reference is omitted in the 'Channel' column. This is not the case for the used montage. Here, signals are computed by the difference of two channels. Therefore, the 'Channel' column contains the two channels whose difference was used as model input.

As expected, all single-channel models perform worse than the multi-channel model. However, some reach up to 60% — not bad for having only $\frac{1}{22}$ of the data available. Few of the single-channel models fail to pass the threshold of 32% that can be achieved by always predicting the majority class. However, this opens the opportunity for further experiments on question **Q6** whether 1D convolutions with residual connections improve model performance.

Table 6.10 gives the single-channel results with channel-wise 1D convolution and residual connection. In an additional column, it shows the difference in accuracy of the compared models. Differences are generally small. The T8-A2 model fails in four cases less and improves good models by 0.5%. On the other hand P7-NO1 fails in 7 instead of in 3 runs.

Channel	# Failing models	Avg acc. (all models)	Avg acc. (good models)
Fp1 – F7	0	0.5270	0.5270
F7 – T7	0	0.6074	0.6074
T7 – P7	0	0.5377	0.5377
P7 – O1	3	0.4280	0.4374
Fp2 – F8	0	0.6044	0.6044
F8 – T8	0	0.5788	0.5788
T8 – P8	0	0.6004	0.6004
P8 – O2	2	0.5095	0.5180
A1 – T7	0	0.4690	0.4690
T7 – C3	0	0.5576	0.5576
C3 – CZ	1	0.4902	0.4941
CZ – C4	1	0.5169	0.5211
C4 – T8	0	0.5771	0.5771
T8 – A2	6	0.4005	0.4137
Fp1 – F3	0	0.4980	0.4980
F3 – C3	0	0.5162	0.5162
C3 – P3	1	0.4911	0.4947
P3 – O1	1	0.4897	0.4932
Fp2 – F4	0	0.5768	0.5768
F4 – C4	0	0.5528	0.5528
C4 – P4	0	0.5429	0.5429
P4 – O2	0	0.5429	0.5429

TABLE 6.9: Single-Channel results, for residual network no 1D conv.

Answer (**Q6**, p. 61): For this data set for single channels, there is no significant difference between models with or without channel-wise convolutions with residual connection.

Finally, I am interested in the question for which brain area the predictions are more accurate (**Q8**). Here, this means which channels are best to distinguish noise. Although Tables 6.9 and 6.10 formally already answer this question, Figure 6.10 visualized this information. Figure 6.9 clarifies the channel locations on the scalp. The data generated by the difference of the electrodes are the inputs for the single-channel models. Thus, each model corresponds to an edge in the graph, for which the result is depicted in Figure 6.10. On average, channels on the right hemisphere perform more than 3.5% better than on the left. Models that use the ear channels give bad predictions, with less than 50% accuracy. On the other hand, channels T7, T8 and Fp2 seem to be good noise detectors, with the best models surpassing 60%. To check whether T7 and T8 are actually detecting chewing better, and Fp2 detects eye movements better, I look at the confusion matrices in Figure 6.11. Each one represents the models with highest accuracy from 50 runs. Indeed, the model that uses Fp2 is at least 9% better at detecting eye movements. Chewing is well detectable by all three single-channel models, even compared to the multi-channel model from Figure 6.8 (p.159). In contrast to the expecta-

Channel	# failing models	avg acc. (all models)	avg acc. (good models)	difference (good models)
Fp1 – F7	$0(\pm 0)$	0.5334	0.5334	$+0.064$
F7 – T7	$0(\pm 0)$	0.6086	0.6086	$+0.012$
T7 – P7	$0(\pm 0)$	0.5544	0.5544	$+0.167$
P7 – O1	$7(+4)$	0.4175	0.4383	$+0.009$
Fp2 – F8	$0(\pm 0)$	0.5858	0.5858	-0.186
F8 – T8	$0(\pm 0)$	0.5909	0.5909	$+0.171$
T8 – P8	$0(\pm 0)$	0.6004	0.6004	$+0.000$
P8 – O2	$1(-1)$	0.5240	0.5286	$+0.106$
A1 – T7	$0(\pm 0)$	0.4723	0.4723	$+0.033$
T7 – C3	$0(\pm 0)$	0.5537	0.5537	-0.039
C3 – CZ	$1(\pm 0)$	0.4568	0.4600	-0.341
CZ – C4	$2(+1)$	0.5107	0.5188	-0.023
C4 – T8	$0(\pm 0)$	0.5580	0.5580	-0.191
T8 – A2	$2(-4)$	0.4567	0.4640	$+0.503$
Fp1 – F3	$0(\pm 0)$	0.5202	0.5202	$+0.222$
F3 – C3	$0(\pm 0)$	0.5197	0.5197	$+0.035$
C3 – P3	$0(-1)$	0.4705	0.4705	-0.242
P3 – O1	$0(-1)$	0.5046	0.5046	$+0.114$
Fp2 – F4	$0(\pm 0)$	0.5897	0.5897	$+0.129$
F4 – C4	$0(\pm 0)$	0.5286	0.5286	-0.242
C4 – P4	$0(\pm 0)$	0.5321	0.5321	-0.108
P4 – O2	$1(+1)$	0.5234	0.5276	-0.153
total	(-1)			$+0.040$

TABLE 6.10: Single-Channel results, for residual network with 1D convolutions and residual connection within it. For better comparison, information is added in each row: The improvement in the number of failing models and the difference in the average accuracy compared to Table 6.9

tions, the $T7 - F7$ model is worst at detecting chewing, but is the best for the *null* class. Of the three, $T8 - P8$ is best at detecting muscle artifacts and electrode pop, although it must be noted that the overall true positive rates (60% and 18%) are still low.

But these results should be interpreted with respect to Table 6.11, which shows the number of seconds of noise present in the single channels and the one from all channels. Given that the labels for chewing coincide means that the model has the chance to learn this class properly. This chance is smaller for *elpp* which is present in less than 25% of the actual time, or eye movement, which is not visible in the $T8 - P8$ channel at all. Knowing that the model classifies eye movement correctly in 50% of the cases is very surprising. This implies that although the label for eye movement is *never* set for this channel, there are indicators in the data that are visible for the machine. Given that *elpp* is visible in channel $Fp2 - F8$ in only 212 seconds, but it is labeled to be there in over 2500 seconds clarifies that this model has difficulties with this class. One would expect only 12% as highest achievable true positive rate,

Channel	eyem	chew	elpp	musc
F7 – T7	419.60	2717.93	280.36	3055.57
Fp2 – F8	6862.30	2689.56	212.49	2218.83
T8 – P8	0.00	2726.44	519.49	2716.65
all	6929.58	2741.19	2527.64	4799.00
avg. right hemisphere	1320.41	2602.94	515.69	2201.99
avg. left hemisphere	1322.27	2575.79	594.92	2036.82

TABLE 6.11: Short summary of time in seconds per class and channel

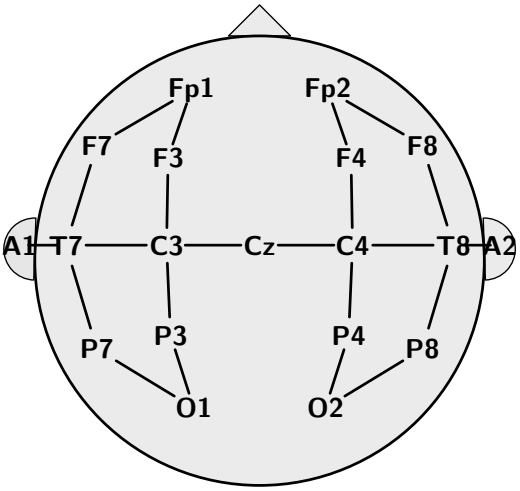


FIGURE 6.9: Transverse central parietal (TCP) montage

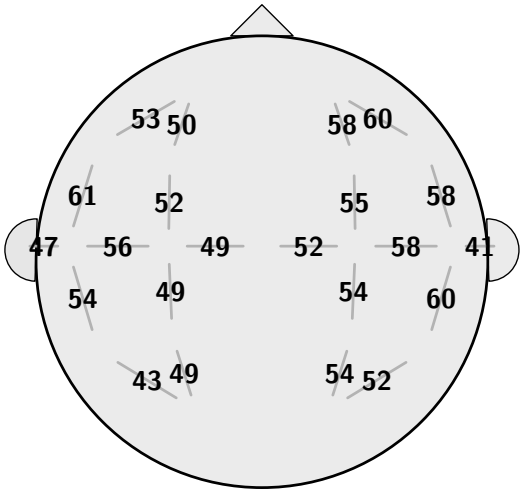


FIGURE 6.10: Average accuracies (in %) of the res model without cw1Dc

so reaching 2% is not much worse than expected. Also muscle movement is with more than 50% sometimes above the expected maximal value.

> **Answer** (**Q8**, p. 71): The right hemisphere seems to be better suited to detect noise. One might speculate that the right hemisphere contains more noise, but this is not supported by the information on the labels for single channels from Table 6.11. The expected effect shows up: channels near the eyes and temples are on average better noise detectors than channels in the central, parietal or occipital regions. When an ear channel is involved, the models perform significantly worse. This is expected, as they do not measure data at the scalp, but at the earlobe. I assume that this is the reason why they are normally used as reference electrodes. The detection of chewing works very well. It is the class with the best true positive rate with more than 85%. Considering that single channel models do not get the labels for their channel, but the ones for all channels, they perform extraordinarily well.

Aggregation

The earlier used aggregation methods rely on the previous knowledge that all snippets of one subject belong to the same class. This is not the case for the given data set. Thus, it makes no sense to perform aggregations here.

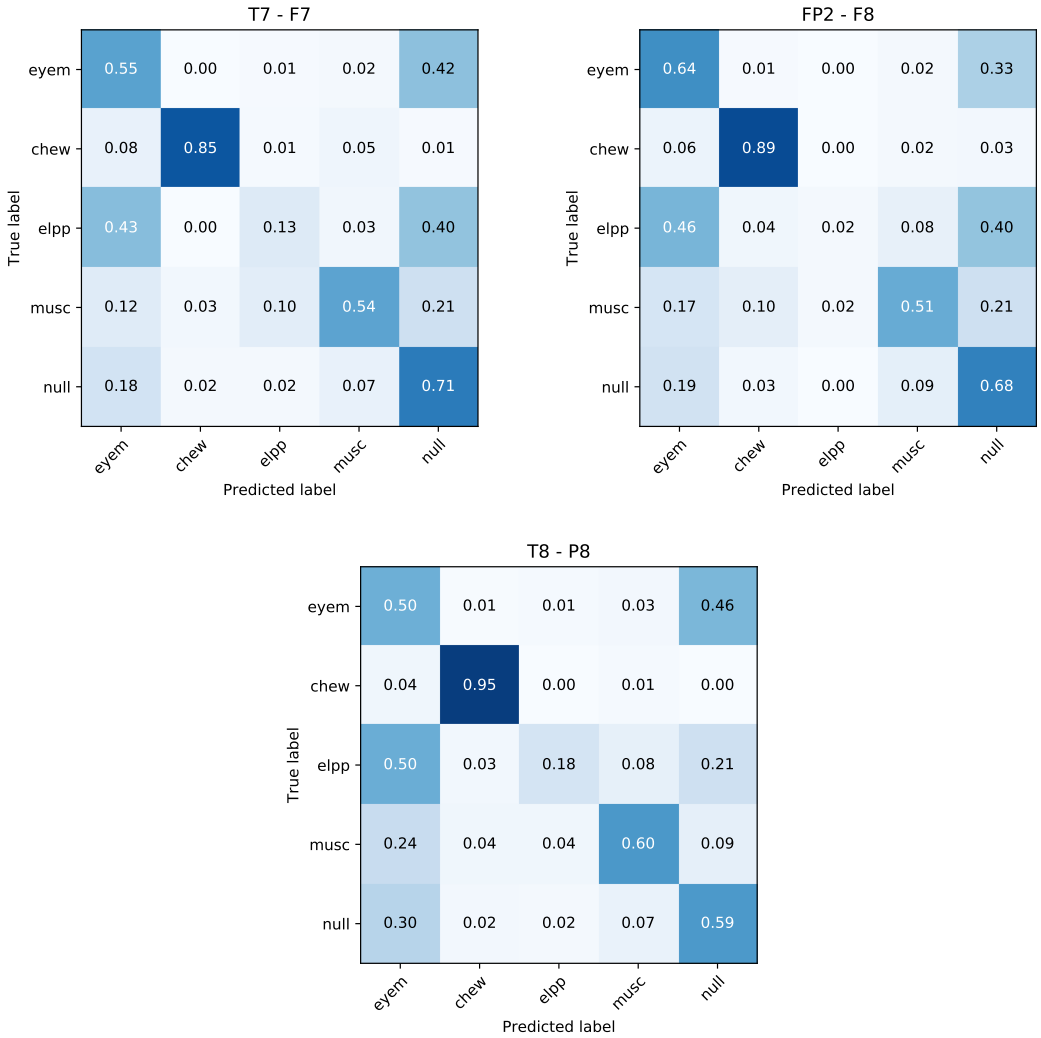


FIGURE 6.11: Confusion matrices for the best models with input channels T7 - F7, Fp2 - F8, T8 - P8

Chapter 7

Conclusion

Over the c(o)urse of this work, I have investigated many questions from computer science and neuro-science. While some answers can be given more generally, others depend on the given task or data set. In the following, I list all questions including their respective answers. I group them to conclude the results first on aggregations in practice, then on different model variations and finally on specific tasks.

Conclusions on Aggregations in Theory and Practice

I start with a set of questions that focus on aggregation functions, and compare their performance in simulations and real data. On simulated data, as in Chapter 4, the distribution can be defined by the simulation parameters. However, to answer these questions on real world scenarios, it must be known a priori that several snippets belong to the same class. This occurs for example when a training sample is too big to be classified as a whole. Then, the original sample can be subdivided into smaller sub-objects (e.g., snippets of time series, or patches of high resolution images). And the results of the sub-objects need to be aggregated in order to create the prediction for the original. The artifact data set from Chapter 6 does not fulfill this criterion, because it is not a subject or measurement session that belongs to a class, but only specific snippets. And multiple classes may be present for the same subject or measurement session. I also could not find any other publicly available data set on human EEG data with fixed labels for longer time intervals, such that aggregation methods could be applied. Thus, I can only answer the following questions for the smoker data set. Answering questions on the snippets' distribution is difficult, as the ground truth is unknown and thus the distributions can only be studied indirectly, by the results that are obtained by a model. And it can only be assumed that the distributions are similar for other (unknown) models.

Q7 (p. 67) Are there snippets, that are better suited than others? Can the class be seen in every snippet? What is the distribution of the snippet quality? Are there snippets that do not contain any helpful information? Are there snippets that mislead the classifier?

Answer (p. 135): The snippets' distribution is far from trivial. Some snippets seem better suited than others, some give the wrong classes very high probabilities and thus it seems like they can even mislead the classifier. The distributions depend highly on the subject, there is no clear pattern and none of them look similar to those seen in the simulations. The detailed results are given in Appendix C, showing triangle plots of the snippet distributions for each of the measurements.

Q9 (p. 86) Is there a bias towards the corners of the k-class projection simplex, when applying a neural network with a softmax layer to real data?

As the simplex can only be easily visualized for three classes and the artifact data set uses more classes, I can only answer it for the smoker data set.

Answer (p. 135): In the performed experiments, there was a bias towards the corners of the triangles. But in contrast to the simulations of Chapter 4, predictions were often not in the middle of the triangle, but close to its edges. The models seem to have learned criteria to exclude one class instead of criteria that would allow to predict one.

Q10 (p. 88) How do the different aggregations perform compared to each other? Is there a method that consistently outperforms the others?

Answer (p. 101): For simulated data, there is no clear best *general* aggregation function, as it depends on the underlying — and in practice often unknown — distribution of sub-objects. Methods *psum* and *prod* perform best for the tested additive and Dirichlet biases, and are the functions to recommend: for small fractions *psum* tends to be better and for small biases *prod* tends to be better. If neither or both fractions and biases are small, then there still seems to be a tendency that *psum* is better for few classes and many snippets. When proper values or estimates for distribution type and for biases and their parameters are given, then the aggregation function should be chosen accordingly.

Answer (p. 119, 126, 140): The aggregation functions *psum*, *qsum*, *dvote* and *avote* perform similarly in all of the experiments on the real data. *Prod* performs similar in most cases, but sometimes clearly worse and is thus not recommended for real world applications. The aggregation *pmax* showed the most variability in the tests and is thus the least robust function. It seems that models that perform only slightly above the level of guessing benefit the most from applying *pmax* aggregation, while otherwise one of the more robust voting-based or summation-based aggregation functions should be chosen.

Q11 (p. 88) How do other factors like the number of classes, the underlying distribution, the number of sub-objects, the fraction of information-carrying sub-objects and the strength of the bias towards the true class in these sub-objects influence the accuracy?

Answer (p. 101): Numerous simulations with *additive* and *Dirichlet* bias show differing results for varying parameters. The methods *psum* and *prod* perform best for the tested biases, and are the functions to recommend: for small fractions *psum* tends to be better and for small biases *prod* tends to be better. If neither or both fractions and biases are small, then there still seems to be a tendency that *psum* is better for few classes and many snippets. When proper values or estimates for type and for biases and their parameters are given, then the aggregation function should be chosen accordingly.

Answer (p. 140): The only experiments on real data compare the *standard testing* with fewer snippets and *extensive testing* with more snippets. In contrast to the simulation, snippets are *dependent* here. Results were on average slightly better for *extensive* testing, but the effect was smaller than fluctuations due to different network initializations. I assume that due to the dependency of the snippets, more snippets do not provide much additional information and thus only improve the predictions negligibly.

Q12 (p. 93) Can (linear) combinations of well performing aggregation functions preserve the best of both functions?

Answer (p. 101): Most linear combinations of *prod* and *psum* were not able to preserve the best properties of both functions on simulated data. For Dirichlet bias *psum* works extraordinarily well and can even dominate *prod*. Hence, the combination of prod and *psum* with high weight on *psum* (*ppmix2*) is also able to dominate *prod* in this case.

Answer (p. 120): Linear combinations of *prod* and *psum* cannot outperform their individual results, on the real data for multi-channel models. The *pmax* function is the best aggregation function in that case.

Answer (p. 125): For each of the tested combined aggregation functions, there is a channel for which it results in the maximal value reached. However, on average *psum* returns the highest value in most of the cases and also the highest average value, which makes it a solid choice for single-channel models. From the tested combined aggregation functions the *fano* family outperforms *ppmix* and *pqmix*.

Conclusions on Model Variations

The following set of questions has the focus on variants of the processing and used models. However, questions **Q3** and **Q4** are based on the assumption that the network uses several convolutional or residual layers. This is not

the case for the best models for the smoker data set. Thus, they cannot be answered for the best models. Those models that made use of convolutional or residual blocks returned worse results. The following answers to these questions build on experiments performed on the artifact data set.

Q2 (p. 44) Which loss function performs better? The classical combination of softmax and categorical cross-entropy or the cosine loss?

Answer (p. 131) For the smoker data set, the cosine loss seems to be better than the classical cross-entropy. Although the basis model was worse in the multi-channel case, after aggregation there was no clear better model. For the single-channel model on channel Fz the results of the basis model as well as those obtained from aggregation with sliding window sampling were clearly better for cosine loss.

Answer (p. 160) For the artifact data set, the cross-entropy loss yields better results than the cosine loss. The number of failing models is higher, and the average accuracy is lower for the cosine loss.

Q15 (p. 109) Which sampling technique performs better? A sliding window or a random sampling?

Answer (p. 118, 125, 140): The difference is generally very small, even negligible for *extensive* testing. When taking only 200 samples, sliding window performs on average slightly better than random sampling. One reason might be a slightly better generalization due to a higher guaranteed coverage of the training samples.

Q3 (p. 58) Is it sufficient to change the initialization to the identity function instead of using a skip connection?

Answer (p. 158): No. Using custom initialization instead of a skip connection produces similar or worse results.

Q4 (p. 61) Does enhancing convolutional blocks with skip connections improve the performance when performing EEG time series classification?

Answer (p. 157): Yes. Enhancing convolutional blocks with skip connections does increase the performance of the model: First, the number of failing models is reduced to 0. Second, even when only considering the good models, the accuracy improves. But note that callbacks that reduce the learning rate on plateaus and apply early stopping are necessary to prevent overfitting.

Q5 (p. 61) Do channel-wise 1D convolutions at the beginning of a model improve their performance?

Answer (p. 119, 125, 157): Rather not. For the smokers data set, results were better *without* 1D convolutions. In some cases for convolutional

models the results vary; some are better, some worse with channel-wise 1D convolutions. Results on the artifact data set differed only marginally.

Q6 (p. 61) Is it helpful to use skip connections in the channel-wise 1D convolutions?

Answer (p. 119, 125, 158, 161): Unclear. The network with skip connections performed worse only for the smoker data set with the multi-channel model. In the multi-channel case of the EEG artifact data set, the channel-wise 1D convolutions with skip connections reduced the number of models that were unable to train from 18 to 13.

Task-Specific Conclusions

Q1 (p. 24) What is the effect of preprocessing on the prediction quality?

Answer (p. 130): Yes. Models perform better on the smoker data set, when the data are preprocessed. This effect is significant for single-channel models but is not significant for multi-channel models. The model run on non-preprocessed multi-channel data shows a special side-effect: it performs worse if aggregation is applied.

Q8 (p. 71) Which brain areas are best suited for the prediction of a given task?

Answer (p. 124): The best brain area for distinguishing craving smokers, non-craving smokers and non-smokers is the central area of the head. The best model used data from the frontal central (Fz) channel, but also models which use data from the frontal left area (F7, F3) perform well.

Answer (p. 163): The right hemisphere seems to be better suited to detect noise. One might speculate that the right hemisphere contains more noise, but this is not supported by the information on the given labels for single channels. The expected effect shows up: channels near the eyes and temples are on average better noise detectors than channels in the central, parietal or occipital regions. When an ear channel is involved, the models perform significantly worse. This is expected, as they do not measure data at the scalp, but at the earlobe. I assume that this is the reason why they are normally used as reference electrodes. The detection of chewing works very well. It is the class with the best true positive rate with more than 85%. Considering that single channel models do not get the labels for their channel, but the ones for all channels, they perform extraordinarily well.

Q13 (p. 104) Can models distinguish between smokers and non-smokers?

Answer (p. 122): On multichannel data, models are able to distinguish between smokers and non-smokers significantly better than random guessing. If the smokers are not craving, then the models need to aggregate several snippets from the same person (best with *pmax* aggregation) in order to reach a significant level. So there is an effect of *smoking* that allows to differentiate between smokers and non-smokers by their EEG data. Although the predictions are significantly better than random guessing, the models are still not very accurate and there is the possibility that they are not reproducible on new data from the same distribution.

Q14 (p. 104) Can models distinguish between craving smokers and non-craving smokers?

Answer (p. 122): Yes. The multi-channel model and most of the single-channel models are able to successfully distinguish between craving and non-craving smokers better than random guessing. Most of the models reach the level of statistical significance, even when no aggregation is applied. The best models achieve 44% of top1 class-balanced accuracy and even 77% when considering top2 class-balanced accuracy. Although statistical significance is given, the differences between the classes are yet too small to be practically useful.

7.1 Discussion

When the available data is as limited as in the cases presented here, it is especially difficult to achieve statistically significant differences between the tested cases. One option would have been to repeat the tests with non-significant results more frequently. But for consistency over the conducted experiments, I decided to have a fixed number of runs in *all* of my experiments: 100. However, my implementation allows to increase the number of runs (e.g., to 1000), even without the need to re-compute the (first) 100 runs.

The problem is the time the computations need. For example the computations for Table 5.8 took a total of 7 weeks on the GPU. Reasons for this long duration are, besides the generally slow process of training neural networks, that 100 runs were performed for each experiment and a total of 75 experiments were conducted — three variants for each of the 25 channels.

Processing several models in parallel on the same GPU is not supported by the Keras framework, as it blocks the entire card, although it does not necessarily need the entire GPU memory. However, the framework *does* support model graphs with multiple inputs and multiple outputs and it does not require that the graph consists of only one connected component. So by creating a graph that consists of one connected component per channel,

it is possible to process all channels in parallel on one card if the following criteria are fulfilled: It must be assured that the GPU memory suffices. All the channels must be processed in a synchronous way, with identical batch size, learning rate, optimizer, callbacks *etc*. A smaller version of model *423N* is model *422Na*. It uses only 128 neurons in the final dense layer and thus needs only about $\frac{1}{7}$ of the memory, which is small enough to process all 25 channels on one GPU in parallel. An implementation can be found in `models/SC422Na_.py`. Results are however worse than those from the bigger model.

Although *extensive* testing produces better results, testing must be performed on more than 1000 times as many snippets as with *standard* testing, which also needs much more computation time and also more space to store the results. The uncompressed results of one run of one sampling method needs approximately 250 megabytes on the hard disk. The results for the simulated data are saved in compressed form and still need more than 36 gigabytes[1].

Neural Networks and Deep Learning

Many tasks for which classical machine learning approaches failed were in recent years approached and solved by neural networks and deep learning methods. Although these results often build new, better baselines for the studied tasks, they do not offer new *insights*, for example by explaining relevant features. One reason is that neural network models are very powerful on one hand, but hardly explainable on the other hand. They use many abstract computations that are difficult to interpret. Many works try to improve neural networks with better network structures, regularization, activation functions, learning rate, learning rate schedules, optimizers, loss functions, callbacks, etc. This is helpful, but mostly ignores the fact that networks are still hardly explainable and thus this does not enable a better *understanding*, but only offers an enlarged set of possibilities for a model's hyperparameters and often even *prevents* understanding.

A reasonable approach would use neural networks only when the relevant features cannot be found with classical methods and apply the networks to detect and understand these features. They could then be used for evaluations that are based on the knowledge of the underlying relations. However, many of the works in the field of neural networks — even when applied to practical problems — do not even attempt to investigate the features.

It seems that there are still some issues with neural networks, and with the publicly available frameworks, seen by the unintuitive application of batch normalization in the Keras framework, described in Section 2.4.10. Another example is the problem of models that are unable to train properly, when badly initialized (see Section 2.4.9). Although this seems to be a fundamental

[1] In an early experimentation phase, the hard disk ran out of space during experimentation, when results were saved uncompressed.

problem for neural networks and was published already 2016 it is hardly ever reported.

In my experiments batch normalization showed a strange behavior: Models on the smoker data were unable to train properly when batch normalization was applied, but for the artifact data this effect was inverted; training worked well with batch normalization, but did not without it. But as the Keras framework [23] has known issues on this topic [149], I did not perform further investigations. As I had already performed many experiments, with Keras, so that it was practically not possible to switch to another framework like PyTorch [111]. However, it is disturbing to know that there are issues of this severity in the widely used Keras framework.

Another problem of neural networks is that the number of possible hyperparameter combinations has reached astronomic dimensions. And while more and more papers investigate the effect of the change of few hyperparameters, and often advertise their own new variations, there is a high (and growing) demand for reliable works that allow insights on which combinations of hyperparameters should be tested in practice, for unknown data sets.

Notes on the Comparability of Studies

One of the huge problems of neuro-science is the comparability and reproducibility of performed studies. Therefore it is mandatory to base the research on data, that is publicly available. In practice this brings many problems: Participants need to be found, who volunteer to participate in science. Since the taken data is individual-specific, very strict legal rules have to be followed in order to protect the individuals rights. An ethics committee needs to approve the study before measurements can begin.

Over the course of this work, I was looking for public EEG data sets from humans and found only few sources. The Temple University Hospital was a great source for the data in Chapter 6. However, although the data is publicly available, there are still many problems for data analysts when trying to compare studies, even if they are based on the same data set: The TUH EEG data uses *European Data Format* (.edf), which is not natively available in many data analysis tools (e.g. KNIME [12]). Fortunately, the Temple University Hospital offers a sample python script that reads the data. But this format can easily be misread by automatic tools, for example by mixing the channel order, if they are not explicitly set in the correct order. Further, the data are in a raw format, and are not processed at all.

Thus researchers have to build their own preprocessing pipeline, including resampling, re-referencing, detecting noise, removing noise, etc. before the learning algorithms can be applied. Further, there is often no standardized cross-validation process such that although the *data* are publicly available, the results are not comparable. Often, scientist still hesitate to publish their source code, which is understandable, since this would publish every detail

of their work, including all their programming bugs. Nevertheless, this is essential for a scientific progress and should thus always be published as additional material to the respective study[2].

7.2 Summary

In this work I applied several techniques to cope with the typical scenario from medicine: First, the number of measurements to be analyzed is extremely small. Second the relevant features are unknown — it may be that the data do not contain any relevant features for the studied task. Third, the measurements consist of multivariate time series, which makes each measurement huge in terms of size or data dimensionality. On one hand this offers the possibility to apply bootstrapping techniques, which creates many statistically dependent snippets that can be used as training samples. On the other hand, the dependency of these groups of snippets must be respected when the data are split into training and test set. If a stratification is needed, for example because the classes are not balanced, then it becomes NP-hard to create the needed split into training and test set.

I investigated several questions regarding the practical application of aggregation functions, methodological variations which try to cope with the few given measurements and finally with some that focus on specific data sets. In the following, I review and assess the main contributions of my work.

Aggregation in Practice

When the class remains constant over a longer period of time, the predictions of several snippets should be aggregated with as many snippets as possible to improve the prediction accuracy. Which aggregation function should be chosen depends on the underlying snippet distribution, which unfortunately is usually unknown in practice.

I did not find a *general* best aggregation function. The *pmax* aggregation is the most unstable. However, it showed the best results in some cases — especially when the basis model was only slightly better than random guessing. Voting and sum-based functions performed very similar and robust, meaning that the variation in classification accuracies is smaller. Sum and squared sum show no clear trend. The assumption that squaring may be beneficial is not supported by my results. Findings are similar for majority vote and vote with abstention: Abstention seems to occur infrequently, hence the results hardly differ and there is no clear trend visible. If there is no further information given, I recommend to use *psum*.

[2] My source code is available at: https://bit.ly/39NyVYD
I want to thank DongKyu Kim for publishing his source code. It built the basis of my own research in Chapter 6.

Experiments show that the distributions used in the simulations differ from the observations from the smoker data. Thus, future experiments could consider two directions: First, simulations with different parameters to model bias, for example by including a form of bias that would consider subject-specific variations. Second, additional data sets with a fixed class per subject could be investigated to discover indicators that allow to choose the optimal aggregation function for new, unknown data sets.

Model Variations

My experiments on skip connections showed that they are able to improve the performance of convolutional models for EEG time series data. Unfortunately, it did not suffice to use an alternative weight initialization to gain the same effect as a skip connection.

Although channel-wise 1D convolutions had shown promising results in preliminary experiments, this did not hold when tested more elaborately. Sometimes they had a positive, but sometimes also a negative effect. The same holds for added skip connections within these channel-wise 1D convolutions.

Using the cosine loss instead of cross-entropy lead to different results, depending on the data set. For the smoker data set, it was better to use the cosine loss, but it caused more failing models and thus a lower average accuracy on the artifact data. I conclude that I cannot give any advice which loss function to use. Thus, I recommend to try both cross-entropy as well as the cosine loss and investigate further to find out whether one outperforms the other.

From the two tested sampling schemes, random sampling and sliding window, I was able to verify the effect that the sliding window is on average slightly better. This was expected, as sliding window has the higher guaranteed coverage of the available training samples.

Task Specific Findings

For the artifact data set, I found those models that use data of the right hemisphere to perform better. Channels near the eyes and temples are on average better noise detectors. When an ear channel is involved, predictions are worse than average. Chewing is with more than 85% the class with the highest true positive rate. I assume this is because chewing is the class that is present very homogeneously in most of the channels. And this is the case because from all given noise labels, chewing is the one that affects the most areas of the head. Eye movements are mostly seen in the frontal channels and electrode pop artifacts are only visible in few channels.

The preprocessing of the smoker data set has a positive effect on the results. This indicates that the independent component analysis and filtering of noisy channels successfully filtered more noise than relevant brain signals.

The main conclusions about smoking are that there are significant differences in the EEG data for both of the inherent effects, for *craving* and for *smoking*. This is especially remarkable, considering that early statistical evaluations on the effects of smoking and craving on the human EEG had been inconsistent and asked practitioners had claimed it was impossible to solve. These promising findings could build the basis of future research with many implications for the study of addiction in cognitive neuroscience.

However, the accuracy is still low and the variance over measurements is fairly high. Models returned very similar predictions for the only left-handed subject, independent of whether she was craving or not. This indicates that the trained models learn patterns specific for a subject. It might further imply that the handedness has a non-negligible effect. I suggest to collect many more measurements in order to obtain more reliable results and to further investigate the relevance of handedness.

With aggregation functions results improve, but they are still far from accurate. However, with the help of a neuro-scientist I tried to relate model inputs and the respective outputs to find the features relevant for this task. After this failed, I successfully conducted permutation tests in order to assure that the models did not pick up random fluctuations, but characteristic differences in the data from craving smokers, non-craving smokers and non-smokers. However, these characteristic differences do not appear to be strong enough in the given data set to allow a neuro-scientist to recognize useful features.

7.3 Future Work

In the following I suggest a few possibilities to extend the presented work. One approach that could be performed on the same data, would combine several single-channel models and to use them to build a new multi-channel model based on the found single-channel features. I would expect them to outperform the multi-channel models that I created without pre-trained models.

Another idea would be to increase the number of runs performed for my experiments in cases, where the results were inconclusive. It would further be interesting to find out whether the handedness does in fact play a role, or if the left-handed subject was simply an outlier in the smoker data set. Therefore, it would be nice to take new measurements, best with a similar amount of smokers and non-smokers, so that the a grouped validation can be performed without the need to solve an NP-hard splitting problem.

For the artifact data set, it would be helpful to verify the correctness of the given labels and fix the ones that seem to be incorrectly labeled. Further, the preprocessing should be standardized such that the results of different algorithms would be better comparable.

It is possible that the data representation (multivariate time series) is not the optimal choice. For the artifact data set, this was already shown in the work of Roy [122]. Thus one could try different formats, like using a Fourier spectrum as input, or data transformed in a time-frequency-domain by a Wavelet transformation.

My experiments on aggregation can easily be extended: my open source code can be extended to perform experiments with different kinds of bias and new aggregation functions. As addition to the current approach, it could also be helpful to train a classifier to choose the optimal aggregation function automatically, or even to create a new combined aggregation function depending on the given data set's properties.

Finally, it would be nice to test the results of my aggregation functions on more data sets from practice in order to close the gap between the simulations and their application.

Appendix A

Aggregation Domination Graphs

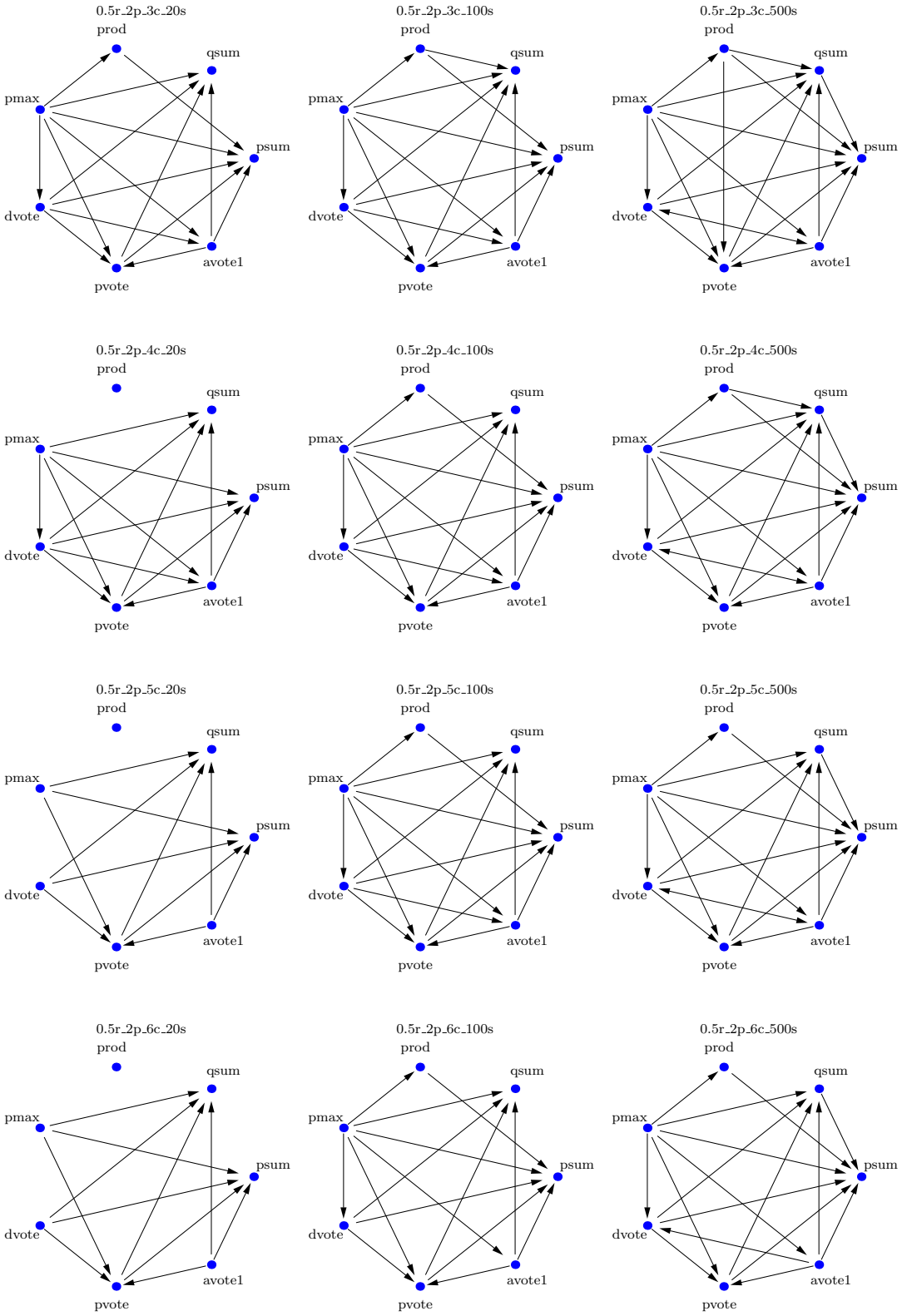


FIGURE A.1: Behavior of the basic aggregation methods for 20, 100 and 500 sub-objects (left to right), 3, 4, 5, 6 classes (top to bottom), Dirichlet bias, $r = 0.5$

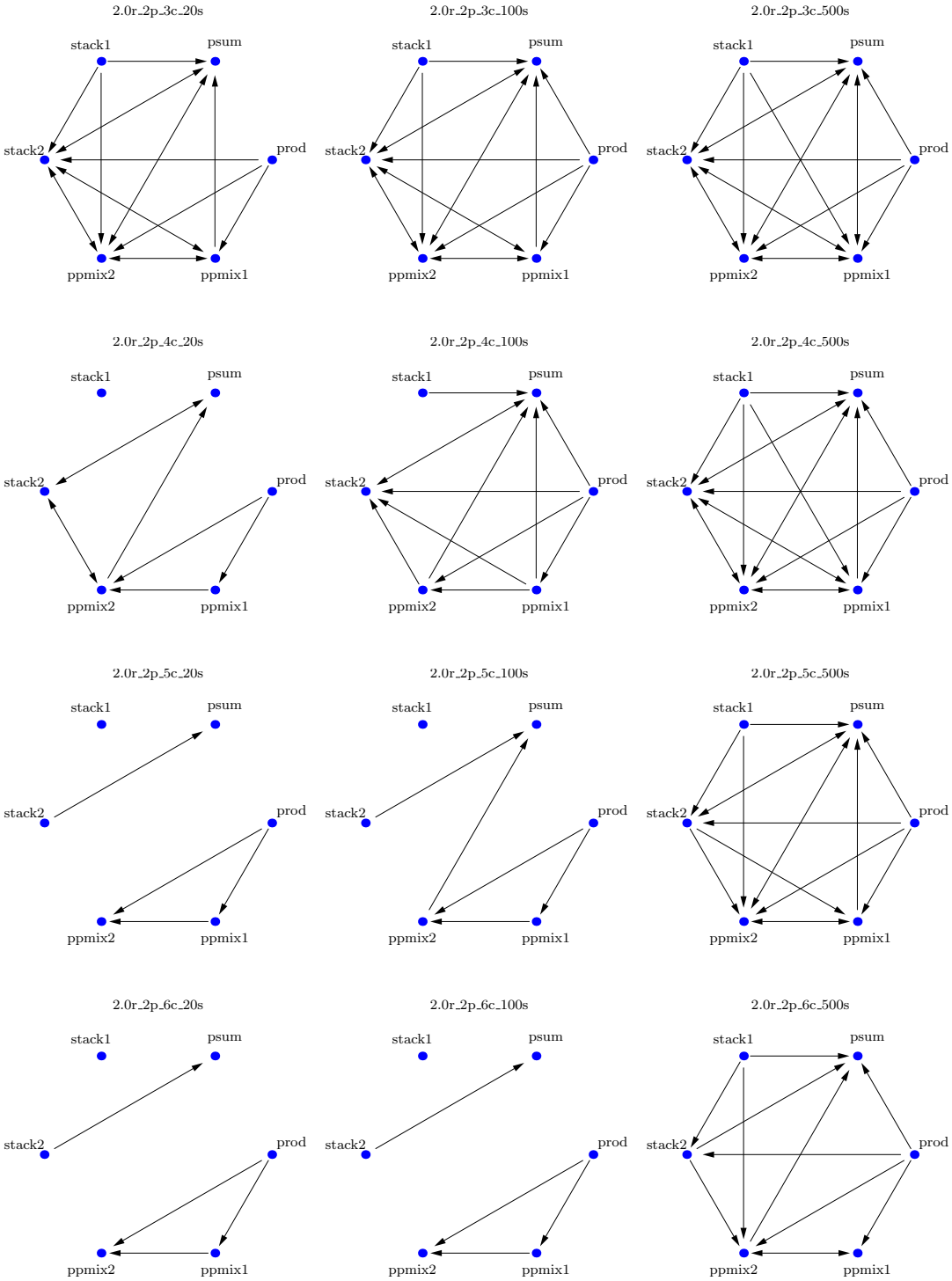


FIGURE A.2: Behavior of prod, psum and four combined aggregation methods for 20, 100 and 500 sub-objects (left to right), 3, 4, 5, 6 classes (top to bottom), Dirichlet bias, $r = 2.0$

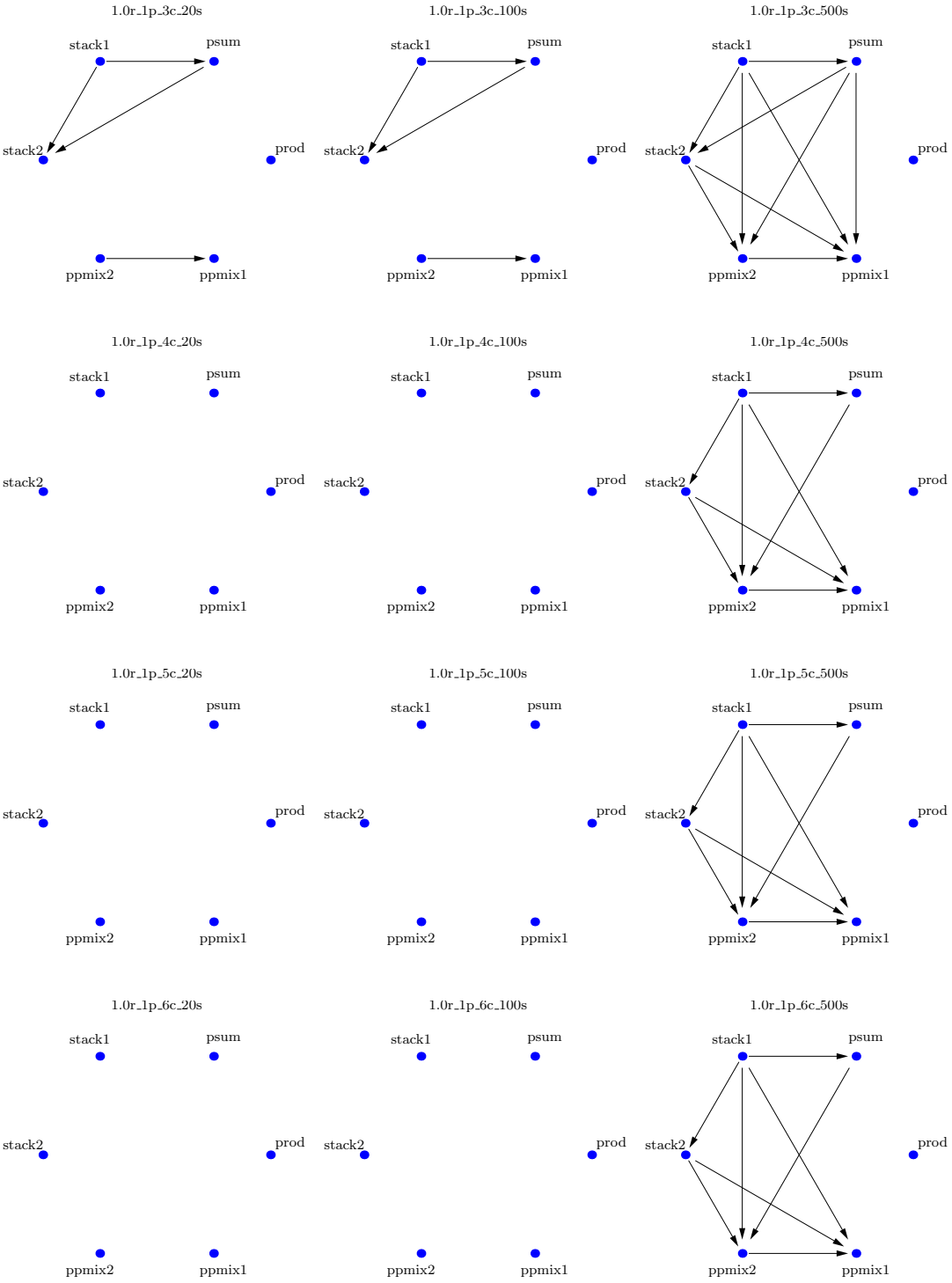


FIGURE A.3: Behavior of the basic aggregation methods for 20, 100 and 500 sub-objects (left to right), 3, 4, 5, 6 classes (top to bottom), additive bias, $r = 1.0$

Appendix B

Additional Results on the Smoker Data

B.1 Results of Variations of the Optimizer

Table B.1 shows the results of models 453, when the Adam optimizer is used instead of Adadelta — those were shown in Table 5.5 (p. 119). The plain model here is slightly better, with 0.3803 instead of 0.3747 as mean value. This has two rather strong effect for the aggregations: First, the performance of all aggregation functions improves about two percent points. Second, pmax does not improve like the others, and performs here on a similar level as the other functions. For random shuffling pmax is even worse than the others. It seems like the chosen optimizer might also influence which aggregation should be chosen to gain best results.

The convolutional and residual versions perform again much worse than the plain model. For those versions pmax is the best performing aggregation. This is surprising, as it did not show up in the plain version. However, the general rule seems to be that pmax is a good choice, if the base models are rather bad.

- Optimizer: Adam
- Learning rate: 0.001 (default)
- Loss function: categorical cross-entropy
- Number of epochs: 100
- Number of batches per epoch: 50
- Batch size: 200
- Total number of parameters: 26,218,575

		Plain		Conv.		Res.	
sampling	agg.	median	mean	median	mean	median	mean
like training	None	0.3750	0.3803	0.3378	0.3431	0.3476	0.3489
random	dvote	0.3888	0.4172	0.3333	0.3666	0.3333	0.3633
random	psum	0.3888	0.4188	0.3333	0.3633	0.3333	0.3627
random	qsum	0.3888	0.4188	0.3333	0.3638	0.3333	0.3611
random	pmax	0.3888	0.3894	0.3888	0.3838	0.3611	0.3694
random	avote1	0.3888	0.4183	0.3333	0.3655	0.3333	0.3655
random	prod	0.3888	0.4055	0.3611	0.3666	0.3611	0.3766
s. window	dvote	0.3888	0.4105	0.3333	0.3650	0.3333	0.3627
s. window	psum	0.3888	0.4122	0.3333	0.3655	0.3333	0.3616
s. window	qsum	0.3888	0.4116	0.3333	0.3650	0.3333	0.3627
s. window	pmax	0.3888	0.4133	0.3888	0.3844	0.3611	0.3794
s. window	avote1	0.3888	0.4094	0.3333	0.3666	0.3333	0.3594
s. window	prod	0.3888	0.4094	0.3888	0.3722	0.3333	0.3744

TABLE B.1: Result overview for the best multichannel neural network model for both sampling schemes, the basic aggregation functions and three variants for the early layers of the models, here performed with Adam as optimizer

B.2 Comparison With and Without Subject Dependency

Figure B.1 shows the effect of subject dependency described in Chapter 5.1 for model 453 (plain) without aggregation. The main effect is that the model's confusion between the craving and non craving classes are reduced by 5.2% and 8.5%. As consequence, all values on the main diagonal are improved. But also the overall prediction rate of the non-smoker class is increased and the confusion between non-craving and non-smoker is decreased. In total, when considering subject dependency, the class-balanced accuracy increases from an average of 33.29% to 37.47%.

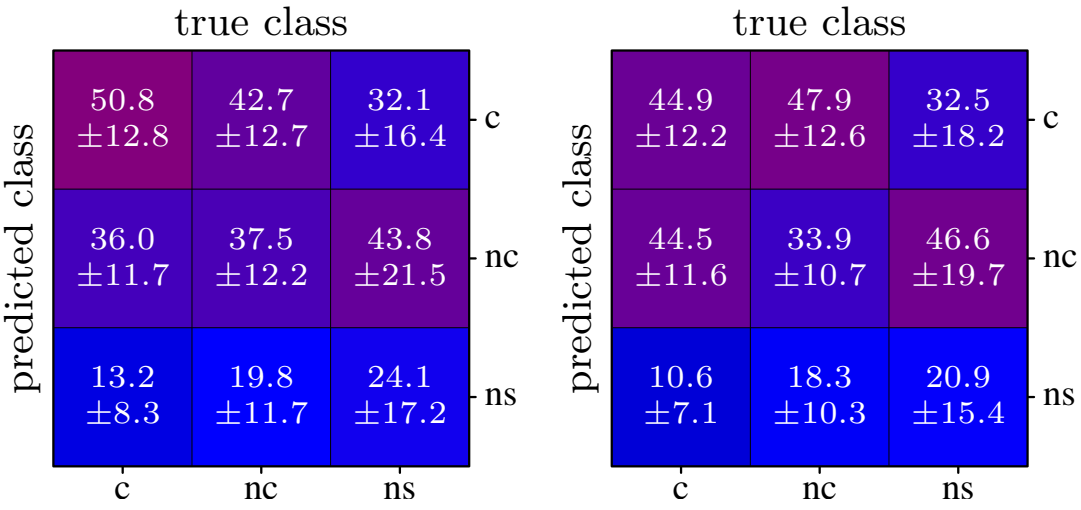
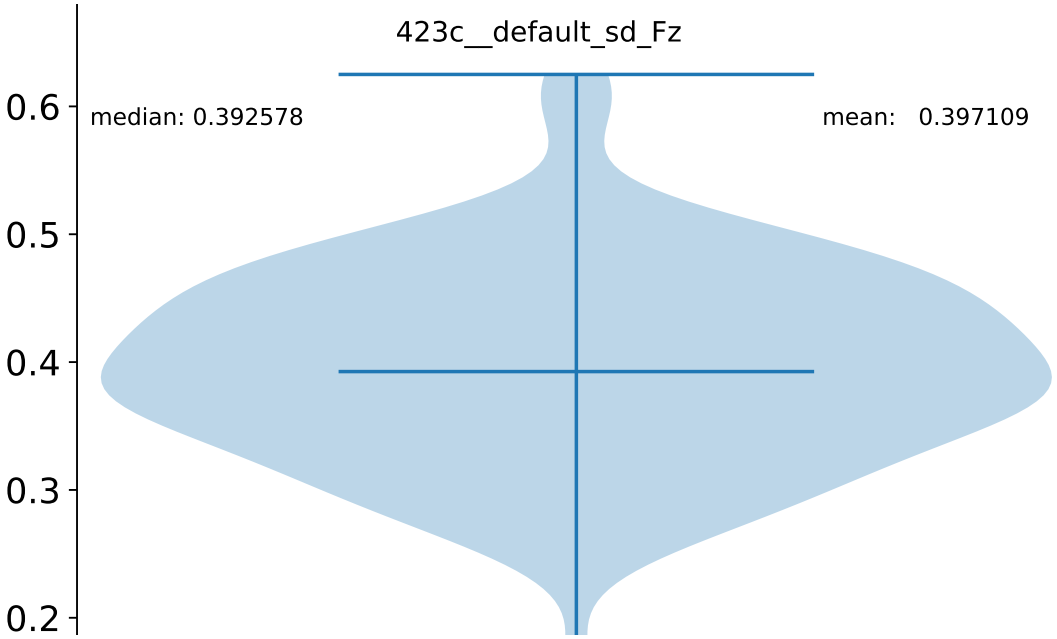
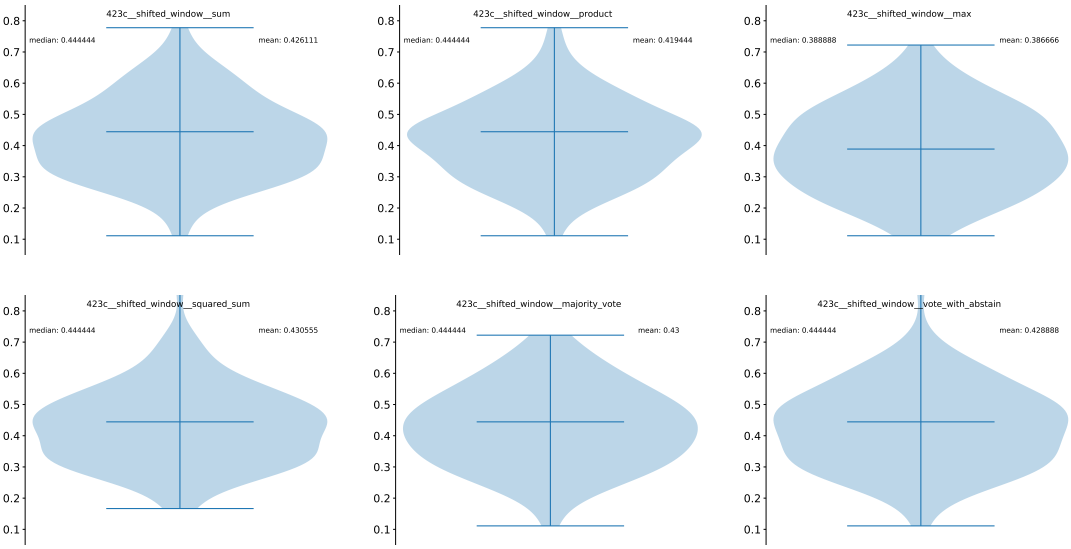


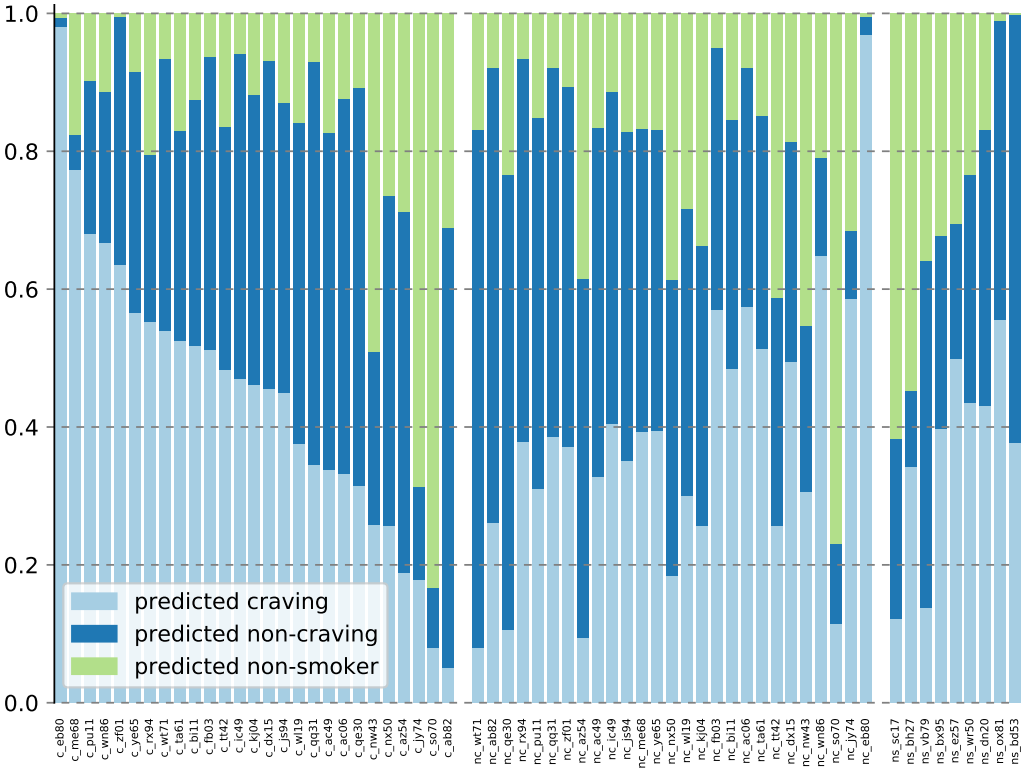
FIGURE B.1: Comparison of confusion matrices with (left) and without (right) subject dependency for model 453 (plain)

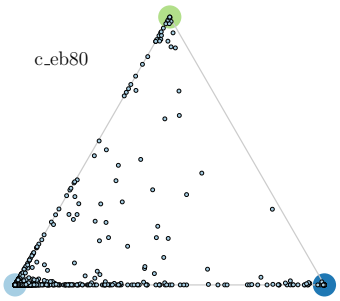
Appendix C

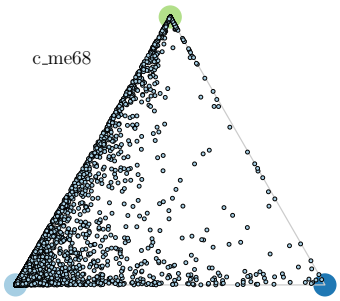
Details on Model *423N* Fz

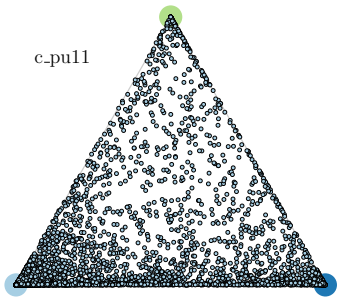


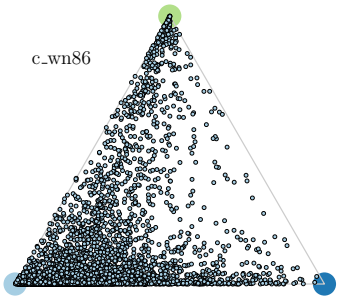


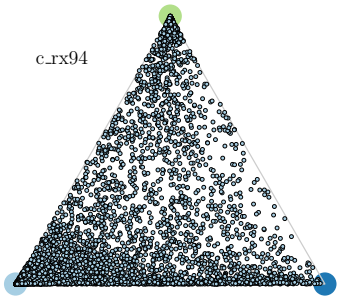

1.0
0.8
0.6
0.4
0.2
0.0
predicted craving
predicted non-craving
predicted non-smoker
c_eb80 c_me68 c_pu11 c_wn86 c_zf01 c_ye65 c_rx94 c_wl71 c_ta61 c_bi11 c_fb03 c_tt42 c_ic49 c_kj04 c_dx15 c_js94 c_wl19 c_qq31 c_ac49 c_ac06 c_qe30 c_nw43 c_nx50 c_ac54 c_jy74 c_so70 c_ab82
nc_wl71 nc_ab82 nc_qe30 nc_rx94 nc_pu11 nc_qq31 nc_zf01 nc_ac54 nc_ac49 nc_ic49 nc_js94 nc_me68 nc_ye65 nc_nx50 nc_wl19 nc_kj04 nc_fb03 nc_bi11 nc_ac06 nc_ta61 nc_tt42 nc_dx15 nc_nw43 nc_wn86 nc_so70 nc_jy74 nc_eb80
ns_sc17 ns_bh27 ns_vb79 ns_bx95 ns_ez57 ns_wr50 ns_dn20 ns_ox81 ns_bd53

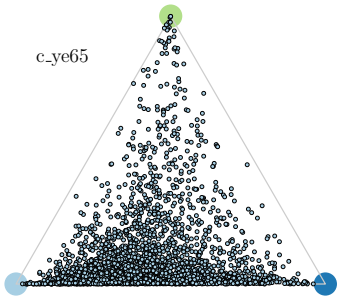

c_eb80

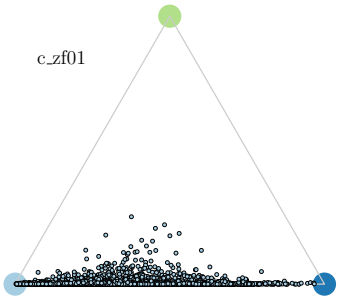

c_me68

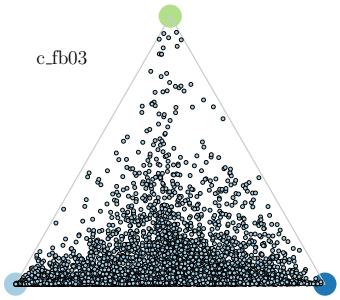

c_pu11

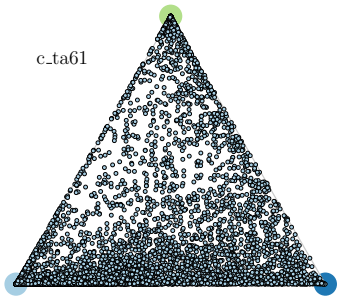

c_wn86

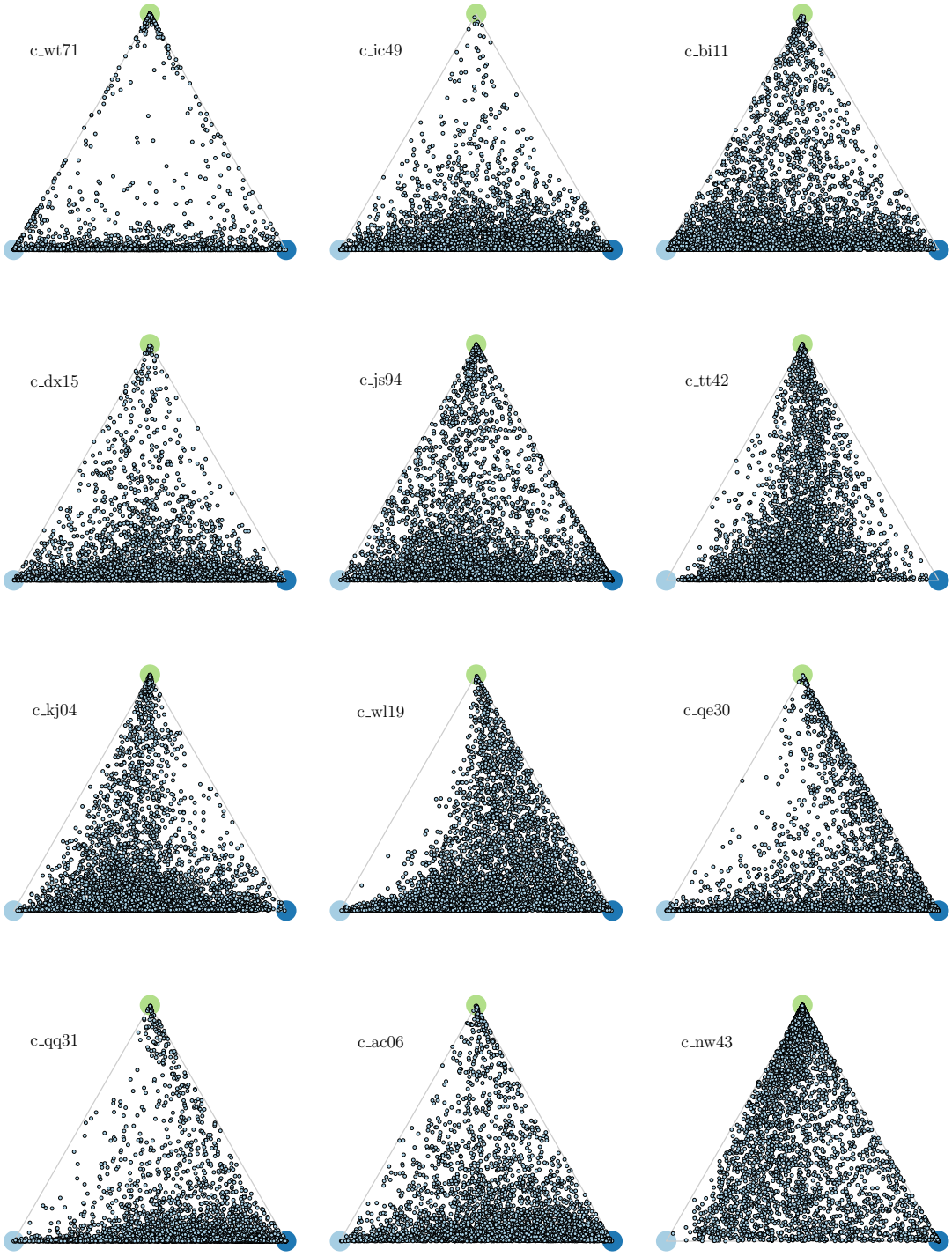

c_rx94

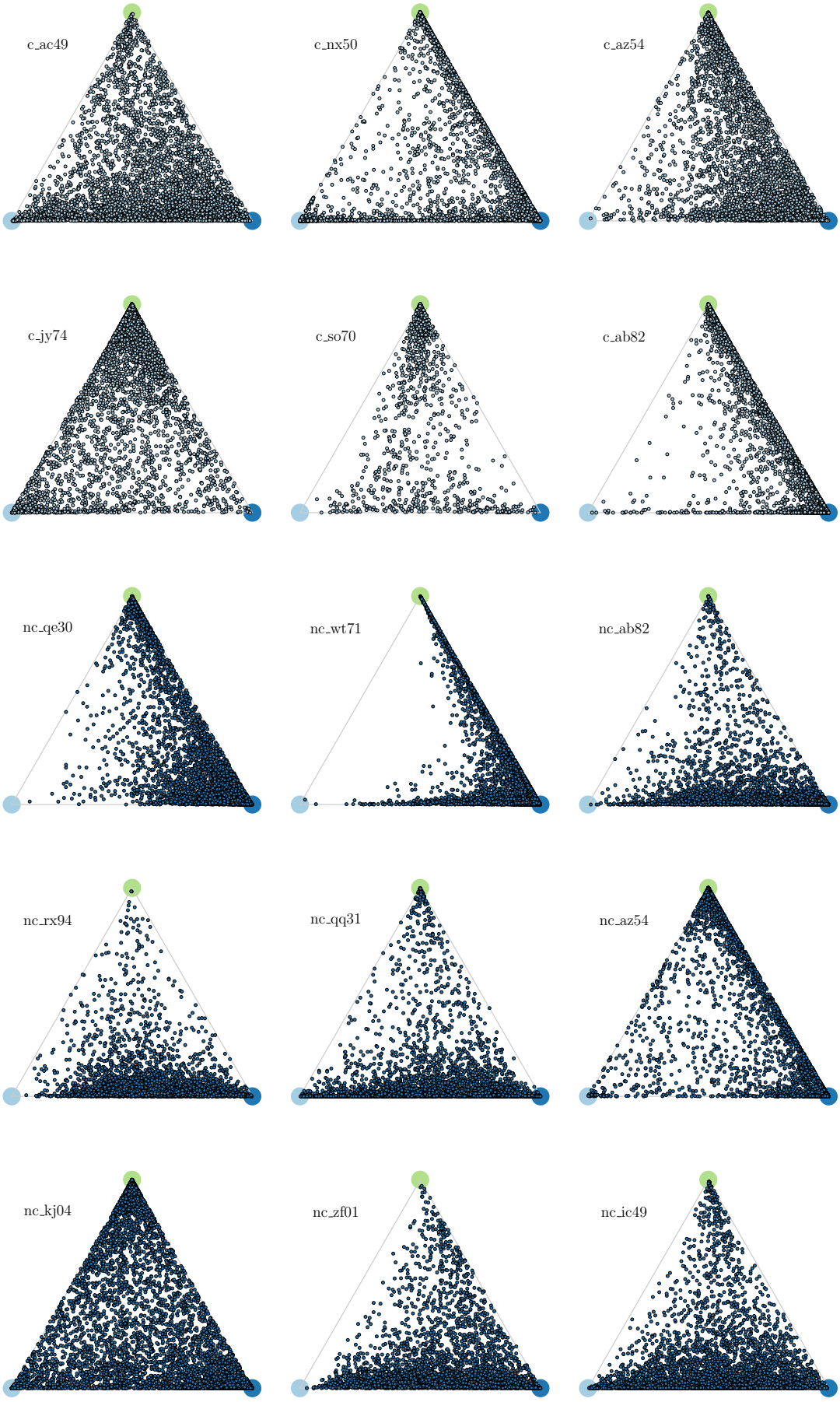

c_ye65

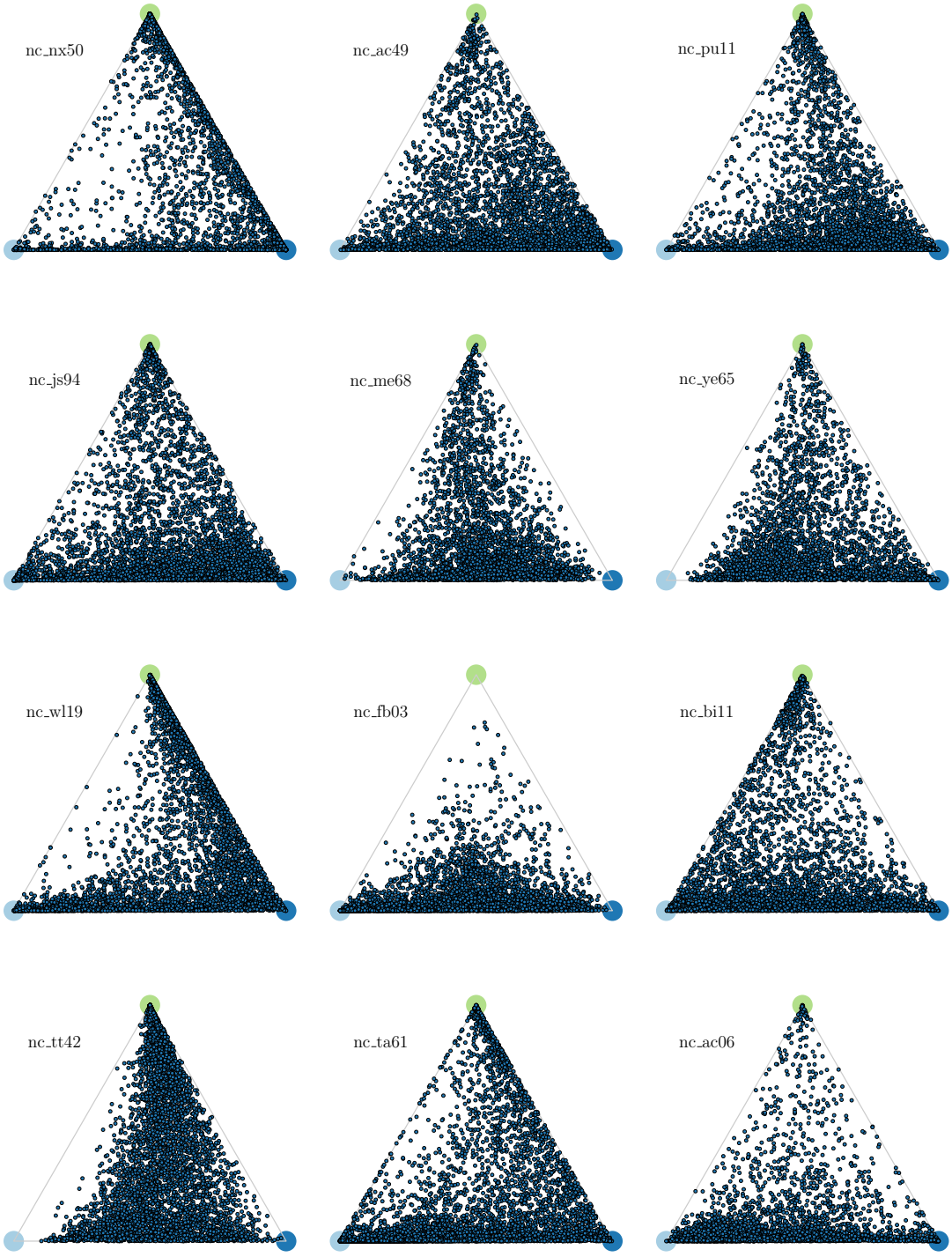

c_zf01

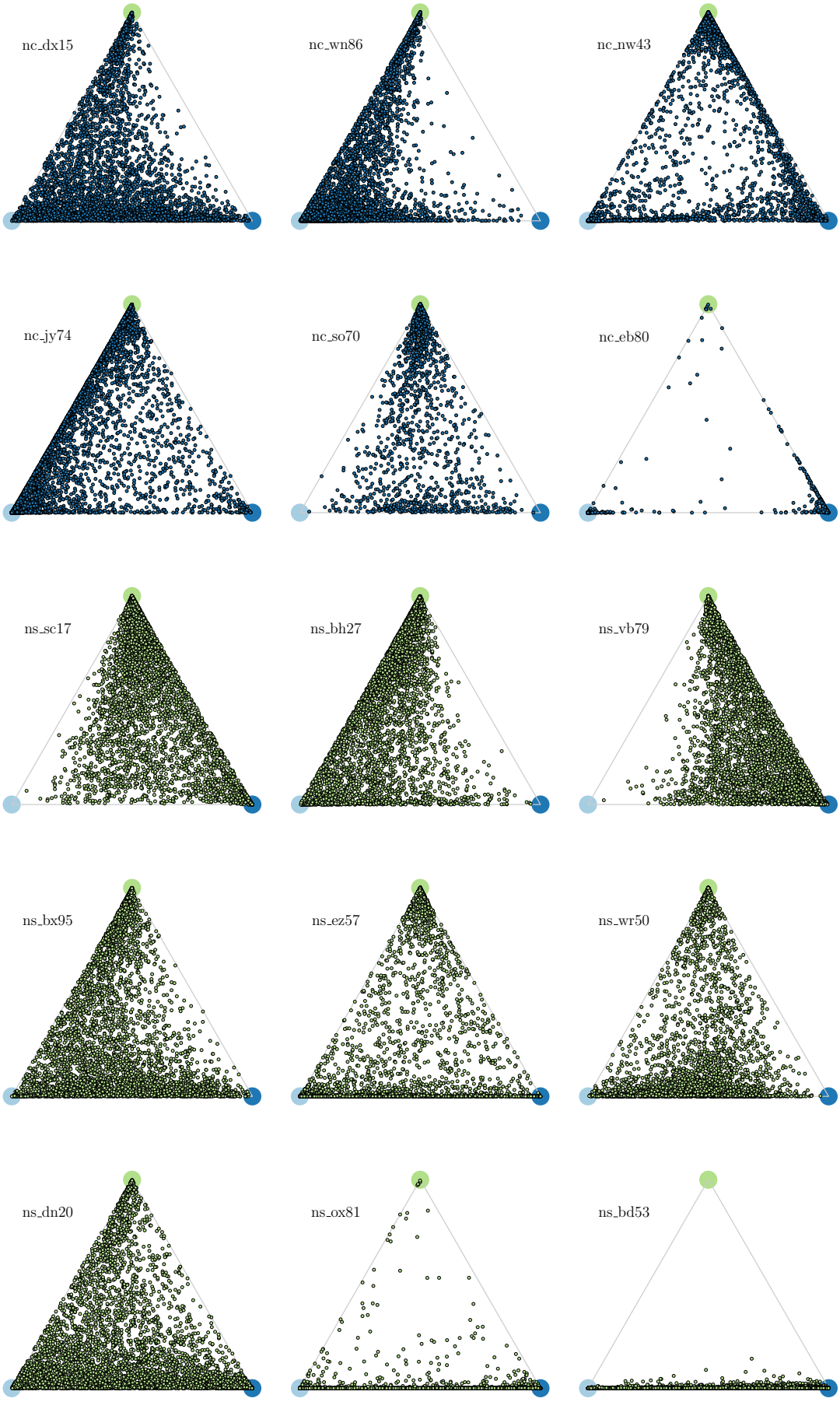

c_fb03

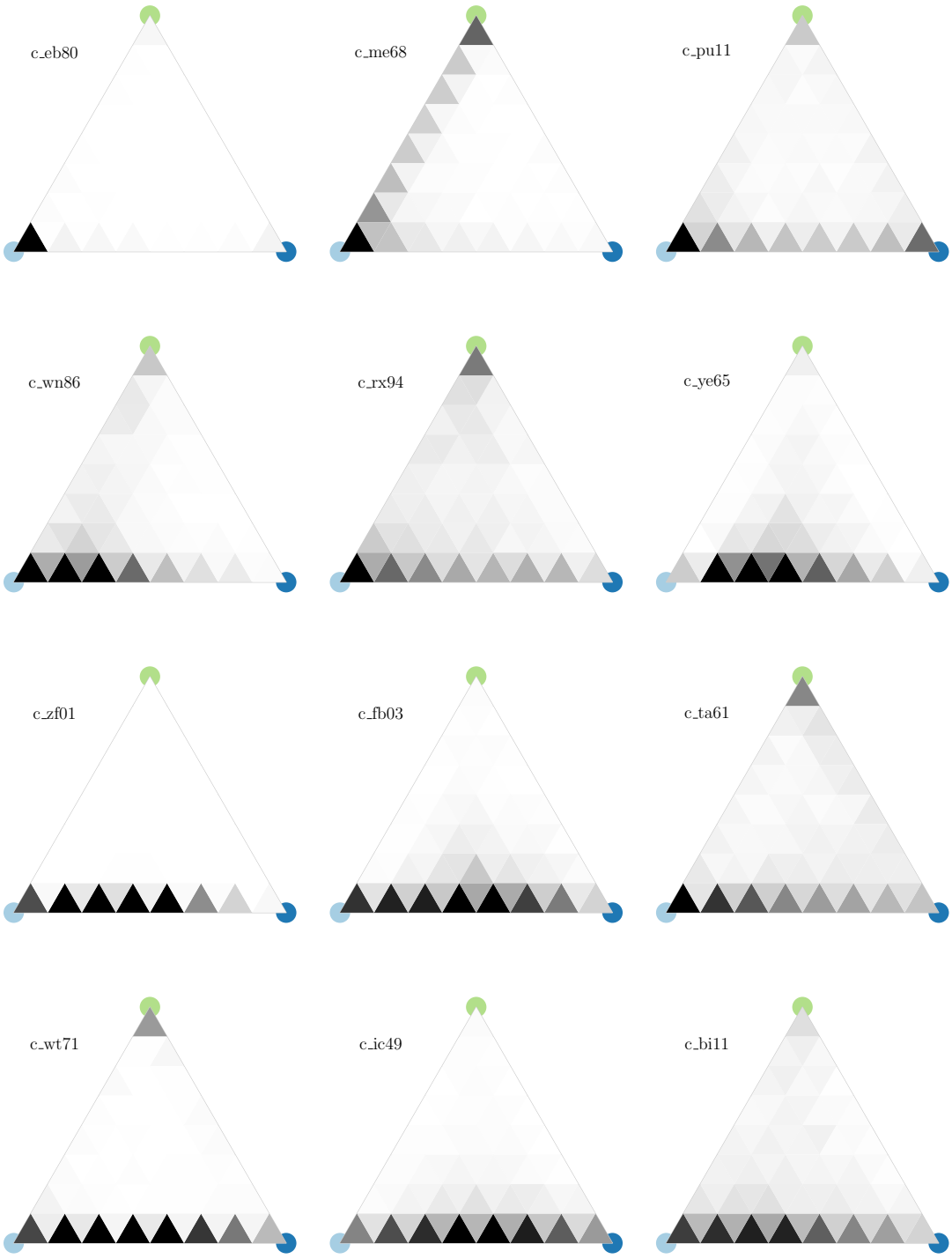

c_ta61

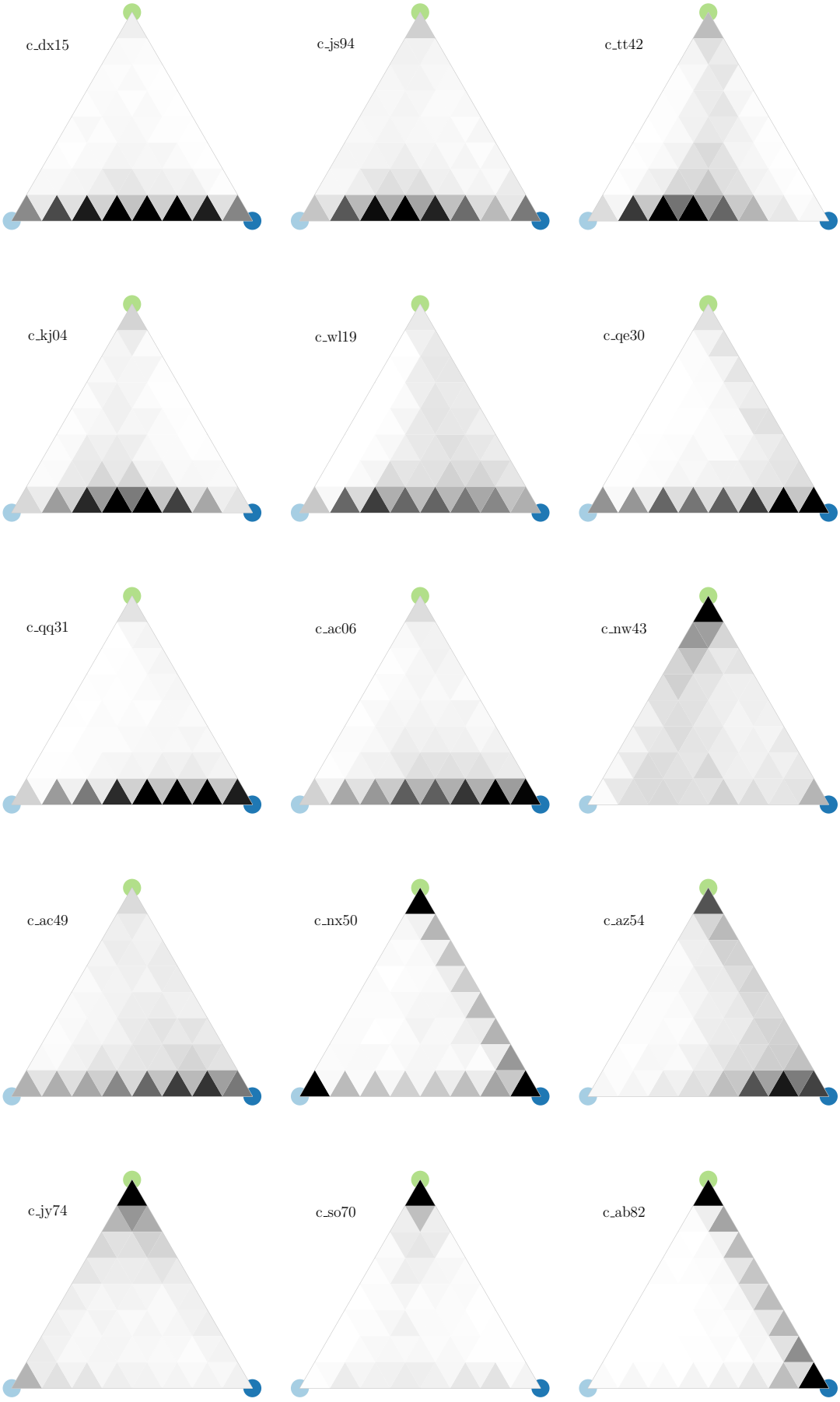

c_wt71
c_ic49
c_bi11
c_dx15
c_js94
c_tt42
c_kj04
c_wl19
c_qe30
c_qq31
c_ac06
c_nw43

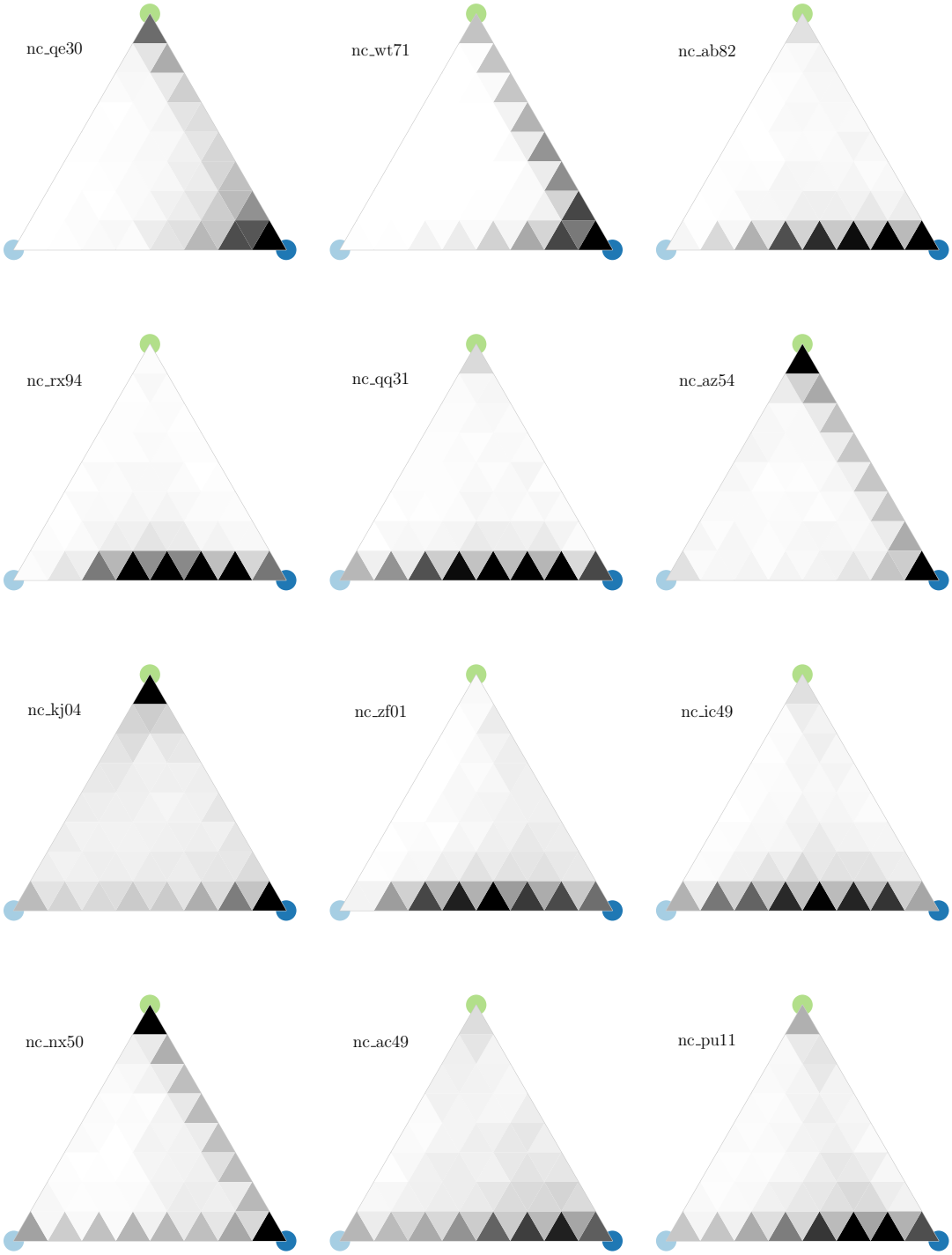

c_ac49
c_nx50
c_az54
c_jy74
c_so70
c_ab82
nc_qe30
nc_wt71
nc_ab82
nc_rx94
nc_qq31
nc_az54
nc_kj04
nc_zf01
nc_ic49

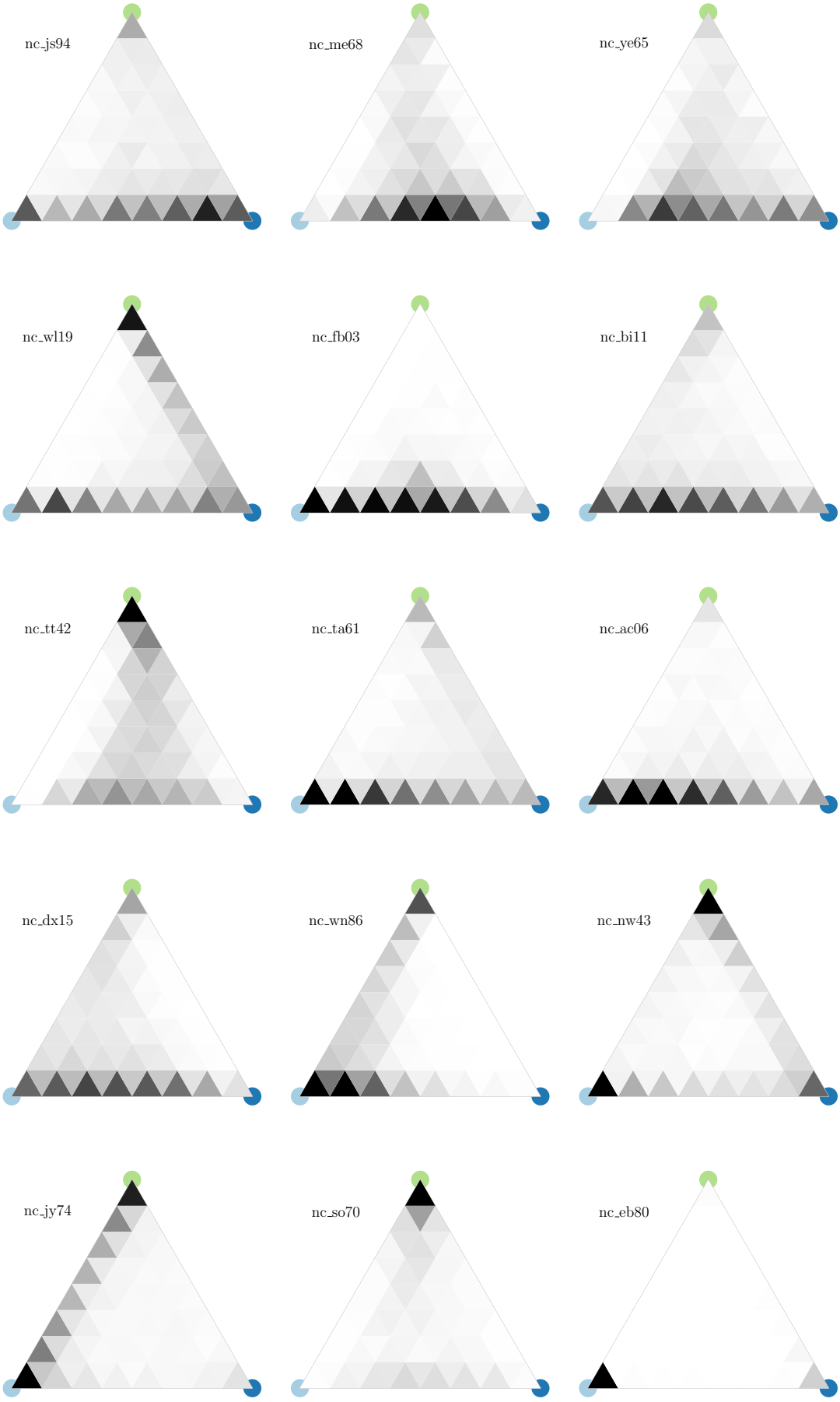

nc_nx50
nc_ac49
nc_pu11
nc_js94
nc_me68
nc_ye65
nc_wl19
nc_fb03
nc_bi11
nc_tt42
nc_ta61
nc_ac06

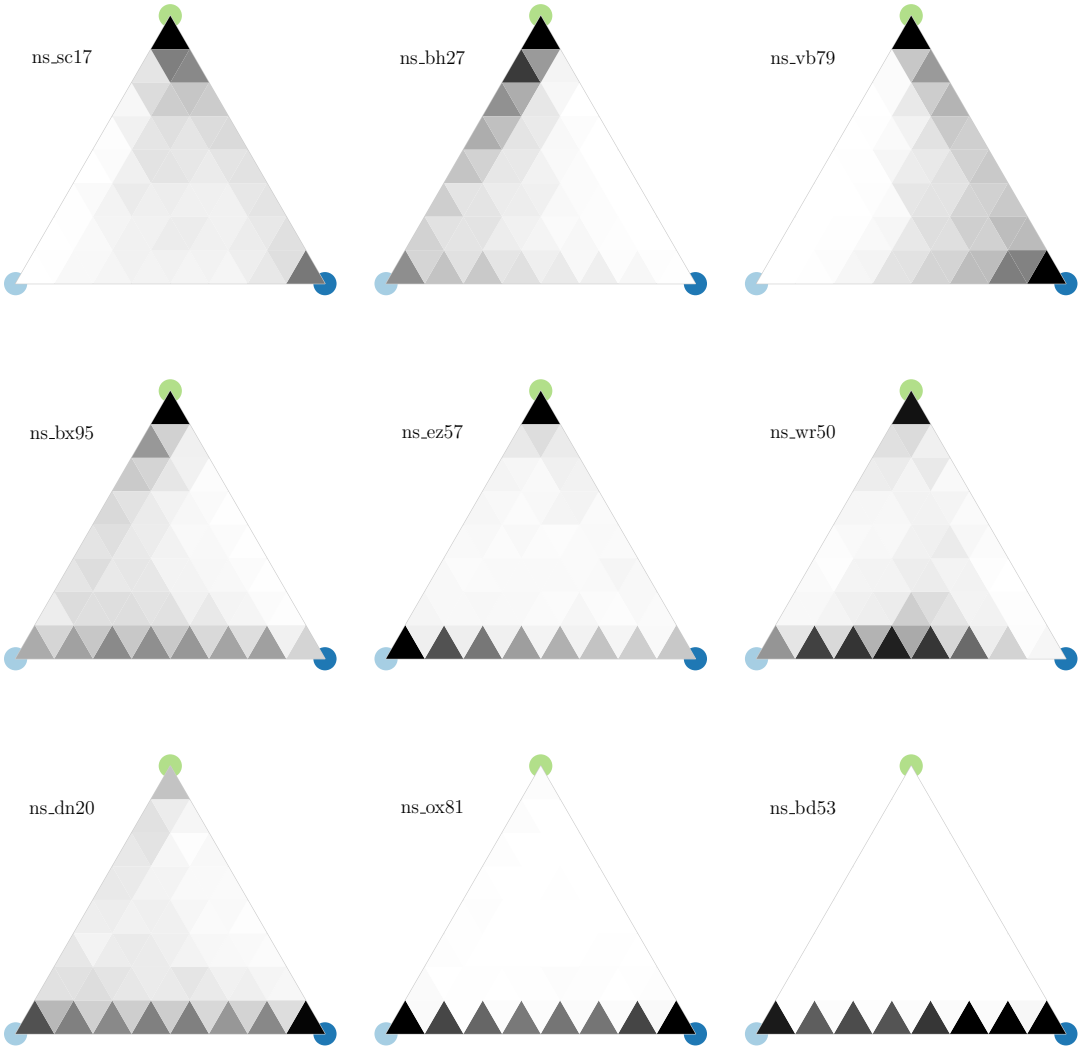

nc_dx15
nc_wn86
nc_nw43
nc_jy74
nc_so70
nc_eb80
ns_sc17
ns_bh27
ns_vb79
ns_bx95
ns_ez57
ns_wr50
ns_dn20
ns_ox81
ns_bd53


c_eb80
c_me68
c_pu11
c_wn86
c_rx94
c_ye65
c_zf01
c_fb03
c_ta61
c_wt71
c_ic49
c_bi11


c_dx15
c_js94
c_tt42
c_kj04
c_wl19
c_qe30
c_qq31
c_ac06
c_nw43
c_ac49
c_nx50
c_az54
c_jy74
c_so70
c_ab82


nc_qe30
nc_wt71
nc_ab82
nc_rx94
nc_qq31
nc_az54
nc_kj04
nc_zf01
nc_ic49
nc_nx50
nc_ac49
nc_pu11


nc_js94
nc_me68
nc_ye65
nc_wl19
nc_fb03
nc_bi11
nc_tt42
nc_ta61
nc_ac06
nc_dx15
nc_wn86
nc_nw43
nc_jy74
nc_so70
nc_eb80


ns_sc17
ns_bh27
ns_vb79
ns_bx95
ns_ez57
ns_wr50
ns_dn20
ns_ox81
ns_bd53

Bibliography

[1] Jayant N Acharya, Abeer J Hani, et al. "American clinical neurophysiology society guideline 3: a proposal for standard montages to be used in clinical EEG". In: *The Neurodiagnostic Journal* 56.4 (2016), pp. 253–260.

[2] Peter Achermann. "EEG analysis applied to sleep". In: *Epileptologie* 26 (2009), pp. 28–33.

[3] Sylvain Arlot, Alain Celisse, et al. "A survey of cross-validation procedures for model selection". In: *Statistics surveys* 4 (2010), pp. 40–79.

[4] Umar Asif, Subhrajit Roy, et al. "Seizurenet: Multi-spectral deep feature learning for seizure type classification". In: *arXiv:1903.03232* (2019).

[5] Zachi I Attia, Paul A Friedman, et al. "Age and sex estimation using artificial intelligence from standard 12-lead ECGs". In: *Circulation: Arrhythmia and Electrophysiology* 12.9 (2019), e007284.

[6] Michael Avendi. *Another look into overfitting*. 2018. URL: `https://medium.com/randomai/another-look-into-over-fitting-33e15b044a5e` (visited on Aug. 21, 2020).

[7] Bjorn Barz and Joachim Denzler. "Deep learning on small datasets without pre-training using cosine loss". In: *The IEEE Winter Conference on Applications of Computer Vision*. 2020, pp. 1371–1380.

[8] Roberto Battiti and Anna Maria Colla. "Democracy in neural nets: Voting schemes for classification". In: *Neural Networks* 7.4 (1994), pp. 691–707. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/0893-6080(94)90046-9`. URL: `http://www.sciencedirect.com/science/article/pii/0893608094900469`.

[9] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[10] James Bergstra, Brent Komer, et al. "Hyperopt: a python library for model selection and hyperparameter optimization". In: *Computational Science & Discovery* 8.1 (2015), p. 014008.

[11] Michael R Berthold, Christian Borgelt, et al. *Guide to intelligent data analysis: how to intelligently make sense of real data*. Springer Science & Business Media, 2010.

[12] Michael R Berthold, Nicolas Cebron, et al. "KNIME-the Konstanz information miner: version 2.0 and beyond". In: *AcM SIGKDD explorations Newsletter* 11.1 (2009), pp. 26–31.

[13] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[14] Christian Borgelt. "Data mining with graphical models". In: (2000).

[15] Christian Borgelt, Matthias Steinbrecher, and Rudolf Kruse. *Graphical Models Representations for Learning*. Wiley Online Library, 2009.

[16] Jeffrey W Britton, Lauren C Frey, et al. *Electroencephalography (EEG): An introductory text and atlas of normal and abnormal findings in adults, children, and infants*. American Epilepsy Society, Chicago, 2016.

[17] B. Brown. "Some characteristic EEG differences between heavy smoker and non-smoker subjects". In: *Neuropsychologia* 6.4 (1968), pp. 381–388. ISSN: 00283932. DOI: 10.1016/0028-3932(68)90010-9. URL: http://linkinghub.elsevier.com/retrieve/pii/0028393268900109.

[18] Jason Brownlee. *Use Early Stopping to Halt the Training of Neural Networks At the Right Time*. 2018. URL: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/ (visited on Aug. 19, 2020).

[19] Raul Gomez Bruballa. *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. 2018. URL: https://gombru.github.io/2018/05/23/cross_entropy_loss/ (visited on Apr. 3, 2019).

[20] Thomas H Budzynski, Helen Kogan Budzynski, et al. *Introduction to quantitative EEG and neurofeedback: Advanced theory and applications*. Academic Press, 2009.

[21] Federico Chella, Vittorio Pizzella, et al. "Impact of the reference choice on scalp EEG connectivity estimation". In: *Journal of neural engineering* 13.3 (2016), p. 036016.

[22] Soo-In Choi and Han-Jeong Hwang. "Effects of Different Re-referencing Methods on Spontaneously Generated Ear-EEG". In: *Frontiers in Neuroscience* 13 (2019), p. 822. ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00822. URL: https://www.frontiersin.org/article/10.3389/fnins.2019.00822.

[23] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015. URL: https://github.com/fchollet/keras.

[24] Robert T Clemen and Robert L Winkler. "Combining probability distributions from experts in risk analysis". In: *Risk analysis* 19.2 (1999), pp. 187–203.

[25] Neural Engineering Data Consortium. *Temple University EEG Corpus*. URL: https://www.isip.piconepress.com/projects/tuh_eeg/ (visited on Aug. 16, 2019).

[26] Neural Engineering Data Consortium. *Temple University EEG Corpus*. URL: https://www.isip.piconepress.com/projects/tuh_eeg/downloads/tuh_eeg_artifact/_AAREADME.txt (visited on Jan. 28, 2020).

[27] James W Cooley and John W Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[28] Dirk De Ridder, Patrick Manning, et al. "The brain, obesity and addiction: an EEG neuroimaging study". In: *Scientific reports* 6 (2016), p. 34122.

[29] J. Deng, W. Dong, et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[30] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[31] Michael Dietz. *Clean and simple Keras implementation of the residual block*. 2018. URL: `https://gist.github.com/mjdietzx/5319e42637ed7ef095d430cb5c5e8c64` (visited on May 25, 2020).

[32] Alexander Dockhorn, Christoph Doell, et al. "A decision heuristic for Monte Carlo tree search doppelkopf agents". In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, pp. 1–8.

[33] Christoph Doell. "Analysis of Electroencephalographic DWT Features for Classification and Regression of Visual Field Charts". MA thesis. Universitaetsplatz 2, 39106 Magdeburg: Otto von Guericke University of Magdeburg, 2013.

[34] Christoph Doell and Christan Borgelt. "Aggregation of Subclassifications: Methods, Tools and Experiments". In: *Proceedings 2019 Symposium Series on Computational Intelligence (SSCI 2019, Xiamen, China)*. 2018, pp. 3131–3138.

[35] Christoph Doell, Sarah Donohue, and Christan Borgelt. "Residual Neural Networks to Distinguish Craving Smokers, Non-craving Smokers and Non-smokers by their EEG Signals". In: *Proceedings 2018 Symposium Series on Computational Intelligence (SSCI 2018, Bengaluru, India)*. 2019, pp. 510–517.

[36] Christoph Doell and Sophie Siebert. "Evaluation of cognitive architectures inspired by cognitive biases". In: *Procedia Computer Science* 88 (2016), pp. 155–162.

[37] Christoph Doell, Pascal Held, et al. "Analysis of a major-accident dataset by Association Rule Mining to minimise unsafe interfaces". In: *13th International Probabilistic Workshop (IPW 2015)*. 2015, pp. 218–230.

[38] Christoph Doell, Sarah Donohue, et al. "Training Neural Networks to Distinguish Craving Smokers, Non-craving Smokers, and Non-smokers". In: *International Symposium on Intelligent Data Analysis*. Springer. 2018, pp. 75–86.

[39] Sarah E Donohue, Marty G Woldorff, et al. "An electrophysiological dissociation of craving and stimulus-dependent attentional capture in smokers". In: *Cognitive, Affective, & Behavioral Neuroscience* 16.6 (2016), pp. 1114–1126.

[40] Sarah E Donohue, Joseph A Harris, et al. "An electrophysiological marker of the desire to quit in smokers". In: *European Journal of Neuroscience* 44.9 (2016), pp. 2735–2741.

[41] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv:1603.07285* (2016).

[42] Saso Dzeroski and Bernard Zenko. "Is Combining Classifiers with Stacking Better than Selecting the Best One?" In: *Machine Learning* 54.3 (Mar. 2004), pp. 255–273. ISSN: 1573-0565. DOI: `10.1023/B:MACH.0000015881.36452.6e`.

[43] Ludwig Fahrmeir, Christian Heumann, et al. *Statistik: Der weg zur Datenanalyse*. Springer-Verlag, 2016.

[44] Ronald A Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2 (1936), pp. 179–188.

[45] Ashley N Gearhardt, William R Corbin, and Kelly D Brownell. "Preliminary validation of the Yale food addiction scale". In: *Appetite* 52.2 (2009), pp. 430–436.

[46] Georgii Gens and Evgenii Levner. "Complexity of approximation algorithms for combinatorial problems: a survey". In: *ACM SIGACT News* 12.3 (1980), pp. 52–65.

[47] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: (1999).

[48] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[50] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv:1412.6572* (2014).

[51] Michel Grabisch, Jean-Luc Marichal, et al. *Aggregation functions*. English. 1. publ. Vol. 127. Cambridge [u.a.]: Cambridge University Press, 2009. ISBN: 0521519268.

[52] Meryem Grabski, H Valerie Curran, et al. "Behavioural tasks sensitive to acute abstinence and predictive of smoking cessation success: a systematic review and meta-analysis". In: *Addiction* 111.12 (2016), pp. 2134–2144.

[53] Alexandre Gramfort, Martin Luessi, et al. "MEG and EEG data analysis with MNE-Python". In: *Frontiers in Neuroscience* 7 (2013).

[54] Carsten Grimm, Christoph Doell, et al. "Network farthest-point diagrams". In: *arXiv preprint arXiv:1304.1909* (2013).

[55] Patrick Grother, Patrick Grother, et al. *Face Recognition Vendor Test (FRVT) Part 2: Identification*. US Department of Commerce, National Institute of Standards and Technology, 2019.

[56] Casper Hansen. *Optimizers Explained - Adam, Momentum and Stochastic Gradient Descent*. 2019. URL: `https://mlfromscratch.com/optimizers-explained/#/` (visited on Aug. 28, 2020).

[57] A Harati, S Lopez, et al. "The TUH EEG CORPUS: A big data resource for automated EEG interpretation". In: *2014 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*. IEEE. 2014, pp. 1–5.

[58] Kaiming He, Xiangyu Zhang, et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[59] Kaiming He, Xiangyu Zhang, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[60] Kaiming He, Xiangyu Zhang, et al. "Identity mappings in deep residual networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 630–645.

[61] Todd F. Heatherton, Lynn T. Kozlowski, et al. "The Fagerström Test for Nicotine Dependence: a revision of the Fagerstrom Tolerance Questionnaire". In: *British Journal of Addiction* 86.9 (1991), pp. 1119–1127. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1360-0443.1991.tb01879.x.

[62] Leandro Hermida. *StratifiedGroupShuffleSplit and StratifiedGroupKFold*. 2020. URL: https://github.com/scikit-learn/scikit-learn/pull/15239 (visited on May 8, 2020).

[63] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: *Diploma, Technische Universität München* 91.1 (1991).

[64] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[65] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251 –257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

[66] https://commons.wikimedia.org/wiki/User:Slashme. *Dopaminergic Pathways*. 2015. URL: https://commons.wikimedia.org/wiki/File:Dopaminergic_pathways.svg (visited on Sept. 2, 2020).

[67] https://commons.wikimedia.org/wiki/User:Ylebru. *eigenfaces*. Credit for publishing to AT&T Laboratories Cambridge. URL: https://upload.wikimedia.org/wikipedia/commons/6/67/Eigenfaces.png (visited on Sept. 15, 2019).

[68] https://en.wikipedia.org/wiki/User:Simpsons_contributor. *File:Valve sobel (1).PNG*. [Online; accessed December 30, 2019]. 2008. URL: https://commons.wikimedia.org/wiki/File:Valve_original_(1).PNG.

[69] https://en.wikipedia.org/wiki/User:Simpsons_contributor. *File:Valve sobel (3).PNG*. [Online; accessed December 30, 2019]. 2008. URL: https://commons.wikimedia.org/wiki/File:Valve_sobel_(3).PNG.

[70] https://www.freepik.com/kjpargeter. *3d head with muscles*. Designed by kjpargeter / Freepik. 2016. URL: https://www.freepik.com/free-photo/3d-head-with-muscles_1020664.htm (visited on July 21, 2020).

[71] Aapo Hyvärinen and Erkki Oja. "Independent component analysis: algorithms and applications". In: *Neural networks* 13.4-5 (2000), pp. 411–430.

[72] Kaggle Inc. *ImageNet Object Localization Challenge*. URL: https://www.kaggle.com/c/imagenet-object-localization-challenge/overview (visited on Sept. 15, 2020).

[73] Kaggle Inc. *Kaggle*. URL: https://www.kaggle.com/datasets (visited on Sept. 23, 2019).

[74] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[75] Kevin Jarrett, Koray Kavukcuoglu, et al. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 2146–2153.

[76] Agnes J Jasinska, Elliot A Stein, et al. "Factors modulating neural reactivity to drug cues in addiction: a survey of human neuroimaging studies". In: *Neuroscience & Biobehavioral Reviews* 38 (2014), pp. 1–16.

[77] Herbert Jasper. "Report of the committee on methods of clinical examination in electroencephalography". In: *Electroencephalogr Clin Neurophysiol* 10 (1958), pp. 370–375.

[78] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[79] DongKyu Kim. "Automatic Artifact Annotator for EEG Waves Using Recurrent and Convolutional Neural Networks". MA thesis. Cooper Union for the advancement of science and art, Albert Nerken School of Engineering, Apr. 28, 2019. URL: `http://ee.cooper.edu/~keene/assets/Thesis_DK_final_final.pdf` (visited on Sept. 12, 2019).

[80] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[81] Verner Knott, Meaghan Cosgrove, et al. "EEG correlates of imagery-induced cigarette craving in male and female smokers". In: *Addictive Behaviors* 33.4 (2008), pp. 616–621. ISSN: 03064603. DOI: `10.1016/j.addbeh.2007.11.006`.

[82] Samuel Kotz, Narayanaswamy Balakrishnan, and Norman L. Johnson. *Continuous Multivariate Distributions — Vol. 1: Models and Applications*. New York, NY, USA: J. Wiley & Sons, 2000.

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

[84] Rudolf Kruse, Christian Borgelt, et al. *Computational Intelligence: A Methodological Introduction*. Springer, 2016.

[85] Ariel Kulik and Hadas Shachnai. "There is no EPTAS for two-dimensional knapsack". In: *Information Processing Letters* 110.16 (2010), pp. 707–710.

[86] Steve Lawrence, C Lee Giles, et al. "Face recognition: A convolutional neural-network approach". In: *IEEE transactions on neural networks* 8.1 (1997), pp. 98–113.

[87] Jean-Yves Le Boudec. *Performance Evaluation of Computer and Communication Systems*. Lausanne: EPFL Press, 2010.

[88] Yann LeCun, Bernhard E Boser, et al. "Handwritten digit recognition with a back-propagation network". In: *Advances in neural information processing systems*. 1990, pp. 396–404.

[89] Honglak Lee, Roger Grosse, et al. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representa-

tions". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 609–616.

[90] Caryn Lerman, Hong Gu, et al. "Large-scale brain network coupling predicts acute nicotine abstinence effects on craving and cognitive function". In: *JAMA psychiatry* 71.5 (2014), pp. 523–530.

[91] HNA Logemann, KBE Böcker, et al. "The effect of the augmentation of cholinergic neurotransmission by nicotine on EEG indices of visuospatial attention". In: *Behavioural brain research* 260 (2014), pp. 67–73.

[92] Silvia Lopez, Aaron Gross, et al. "An analysis of two common reference points for EEGS". In: *2016 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*. IEEE. 2016, pp. 1–5.

[93] Chuck Lorre and Bill Prady. "The Big Bang Theory, Season 04, Episode 21". In: New York, NY, USA: Columbia Broadcasting System (CBS), 2011.

[94] Zhou Lu, Hongming Pu, et al. "The Expressive Power of Neural Networks: A View from the Width". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, et al. Curran Associates, Inc., 2017, pp. 6231–6239. URL: http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf.

[95] Steven J Luck. *An Introduction to the Event-Related Potential Technique (Cognitive Neuroscience)*. A Bradford Book, 2005.

[96] Mufti Mahmud, Mohammed Shamim Kaiser, et al. "Applications of deep learning and reinforcement learning to biological data". In: *IEEE transactions on neural networks and learning systems* 29.6 (2018), pp. 2063–2079.

[97] Rinat Maksutov. *Deep study of a not very deep neural network. Part 4: How to find the right learning rate*. 2018. URL: https://medium.com/@maksutov.rn/deep-study-of-a-not-very-deep-neural-network-part-4-how-to-find-the-right-learning-rate-e06d6da26b2e (visited on Aug. 27, 2020).

[98] Farrokh Mansouri, Arsalan Mir-Moghtadaei, et al. "Development and validation of a 3D-printed neuronavigation headset for therapeutic brain stimulation". In: *Journal of neural engineering* 15.4 (2018), p. 046034.

[99] George Marsaglia and Wai Wan Tsang. "A Simple Method for Generating Gamma Variables". In: *ACM Trans. on Mathematical Software* 26 (3 2000), pp. 363–372.

[100] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[101] Juan M. Montoya, Christoph Doell, and Christian Borgelt. "Wide and Deep Reinforcement Learning Extended for Grid-Based Action Games". In: *Agents and Artificial Intelligence*. Ed. by Jaap van den Herik, Ana Paula Rocha, and Luc Steels. Cham: Springer International Publishing, 2019, pp. 224–245. ISBN: 978-3-030-37494-5.

[102] Raphael Moura, Christoph Doell, et al. "A clustering approach to a major-accident data set: Analysis of key interactions to minimise hu-

man errors". In: *2015 IEEE Symposium Series on Computational Intelligence*. IEEE. 2015, pp. 1838–1843.

[103] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *ICML*. 2010.

[104] Ernst Niedermeyer and FH Lopes da Silva. *Electroencephalography: basic principles, clinical applications, and related fields*. Lippincott Williams & Wilkins, 2005.

[105] Michael A Nielsen. *Neural networks and deep learning*. Vol. 2018. Determination press San Francisco, CA, 2015.

[106] Ernest P Noble. "D2 dopamine receptor gene in psychiatric and neurologic disorders and its phenotypes". In: *American Journal of Medical Genetics Part B: Neuropsychiatric Genetics* 116.1 (2003), pp. 103–125.

[107] Ingram Olkin and Herman Rubin. "Multivariate Beta Distributions and Independence Properties of the Wishart Distribution". In: *The Annals of Mathematical Statistics* 35 (1 1964), pp. 261–269.

[108] Ibrahim Omerhodzic, Samir Avdakovic, et al. "Energy distribution of EEG signals: EEG signal wavelet-neural network classifier". In: *arXiv preprint arXiv:1307.7897* (2013).

[109] Vani Pariyadath, Elliot A Stein, and Thomas J Ross. "Machine learning classification of resting state functional connectivity predicts smoking status". In: *Frontiers in human neuroscience* 8 (2014), p. 425.

[110] R Parra-Hernandez and N Dimopoulos. "Heuristic approaches for solving the multidimensional knapsack problem (mkp)". In: *WSEAS Transactions on Systems* 1.2 (2002), pp. 248–253.

[111] Adam Paszke, Sam Gross, et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[112] Cedrik Pätz. "Analysis of EEG signals for the Detection of Nicotine Dependence". MA thesis. University of Magdeburg, Institute for Intelligent Cooperating Systems, Oct. 9, 2017.

[113] F. Pedregosa, G. Varoquaux, et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[114] François Perrin, J Pernier, et al. "Spherical splines for scalp potential and current density mapping". In: *Electroencephalography and clinical neurophysiology* 72.2 (1989), pp. 184–187.

[115] Andre Perunicic. *Understanding Neural Network Weight Initialization*. 2017. URL: `https://intoli.com/blog/neural-network-initialization/` (visited on Nov. 27, 2019).

[116] Emilio Merlo Pich, Sonia R Pagliusi, et al. "Common neural substrates for the addictive properties of nicotine and cocaine". In: *Science* 275.5296 (1997), pp. 83–86.

[117] Henri Poincaré. *La science et l'hypothèse*. Flammarion, 1902.

[118] Lutz Prechelt. "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[119] David Premack. "Human and animal cognition: Continuity and discontinuity". In: *Proceedings of the National Academy of Sciences* 104.35 (2007), pp. 13861–13867.

[120] Olga Rass, Woo Young Ahn, and Brian F. O'Donnell. "Resting-state EEG, impulsiveness, and personality in daily and nondaily smokers". In: *Clinical Neurophysiology* 127.1 (2016), pp. 409–418. ISSN: 18728952. DOI: 10.1016/j.clinph.2015.05.007. arXiv: 15334406. URL: http://dx.doi.org/10.1016/j.clinph.2015.05.007.

[121] Edward L Reilly. "EEG recording and operation of the apparatus". In: *Electroencephalography* (1982).

[122] Subhrajit Roy. "Machine Learning for removing EEG artifacts: Setting the benchmark". In: *arXiv preprint arXiv:1903.07825* (2019).

[123] Yannick Roy, Hubert Banville, et al. "Deep learning-based electroencephalography analysis: a systematic review". In: *Journal of neural engineering* 16.5 (2019), p. 051001.

[124] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[125] Stuart Russell and Peter Norvig. *Künstliche Intelligenz. Ein moderner Ansatz, 3. ak. Aufl*. 2013.

[126] Hanno Scharr. "Optimal operators in digital image processing". PhD thesis. 2000.

[127] V Schetinin, L Jakaite, et al. "Feature Extraction with GMDH-Type Neural Networks for EEG-Based Person Identification." In: *International journal of neural systems* (2017), p. 1750064.

[128] Robin Tibor Schirrmeister, Jost Tobias Springenberg, et al. "Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG". In: *arXiv:1703.05051* (2017).

[129] Arnold Schönhage and Volker Strassen. "Schnelle Multiplikation grosser Zahlen". In: *Computing* 7.3-4 (1971), pp. 281–292.

[130] Fabrizio Sebastiani. "Machine learning in automated text categorization". In: *ACM computing surveys (CSUR)* 34.1 (2002), pp. 1–47.

[131] Claude E Shannon. "XXII. Programming a computer for playing chess". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41.314 (1950), pp. 256–275.

[132] Claude Elwood Shannon. "A mathematical theory of communication". In: *Bell system technical journal* 27.3 (1948), pp. 379–423.

[133] Thalles Silva. *An illustrative introduction to Fisher's Linear Discriminant*. 2019. URL: https://sthalles.github.io/fisher-linear-discriminant/ (visited on Aug. 12, 2020).

[134] David Silver, Aja Huang, et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[135] David Silver, Julian Schrittwieser, et al. "Mastering the game of go without human knowledge". In: *nature* 550.7676 (2017), pp. 354–359.

[136] Steven Smith. *Digital signal processing: a practical guide for engineers and scientists*. Elsevier, 2013.

[137] Steven W Smith et al. "The scientist and engineer's guide to digital signal processing". In: (1997).

[138] Jost Tobias Springenberg, Alexey Dosovitskiy, et al. "Striving for simplicity: The all convolutional net". In: *arXiv:1412.6806* (2014).

[139] Nitish Srivastava, Geoffrey Hinton, et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[140] Luke E Stoeckel, Xiaoqian J Chai, et al. "Lower gray matter density and functional connectivity in the anterior insula in smokers compared with never smokers". In: *Addiction biology* 21.4 (2016), pp. 972–981.

[141] Samuel Sutton, Margery Braren, et al. "Evoked-potential correlates of stimulus uncertainty". In: *Science* 150.3700 (1965), pp. 1187–1188.

[142] Grzegorz Swirszcz, Wojciech Marian Czarnecki, and Razvan Pascanu. "Local minima in training of neural networks". In: *arXiv:1611.06310* (2016).

[143] Babak A Taheri, Robert T Knight, and Rosemary L Smith. "A dry electrode for EEG recording". In: *Electroencephalography and clinical neurophysiology* 90.5 (1994), pp. 376–383.

[144] Michal Teplan et al. "Fundamentals of EEG measurement". In: *Measurement science review* 2.2 (2002), pp. 1–11.

[145] Matthew A Turk and Alex P Pentland. "Face recognition using eigenfaces". In: *Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition*. IEEE Computer Society. 1991, pp. 586–587.

[146] Francesco Versace, Jennifer A Minnix, et al. "Brain reactivity to emotional, neutral and cigarette-related stimuli in smokers". In: *Addiction biology* 16.2 (2011), pp. 296–307.

[147] Francesco Versace, Jason D Robinson, et al. "Cigarette cues capture smokers' attention: Evidence from event-related potentials". In: *Psychophysiology* 47.3 (2010), pp. 435–441.

[148] Pauli Virtanen, Ralf Gommers, et al. *scipy.signal.resample*. 2017. URL: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.resample.html` (visited on Aug. 3, 2019).

[149] Vasilis Vryniotis. *Align BN trainable behaviour with TF 2.0*. 2020. URL: `https://github.com/keras-team/keras/pull/13892` (visited on July 8, 2020).

[150] Vasilis Vryniotis. *The Batch Normalization layer of Keras is broken*. 2018. URL: `http://blog.datumbox.com/the-batch-normalization-layer-of-keras-is-broken/` (visited on July 8, 2020).

[151] Barbara J Weiland, Amithrupa Sabbineni, et al. "Reduced executive and default network functional connectivity in cigarette smokers". In: *Human brain mapping* 36.3 (2015), pp. 872–882.

[152] Reagan R Wetherill, Hengyi Rao, et al. "Classifying and characterizing nicotine use disorder with high accuracy using machine learning and resting-state fMRI". In: *Addiction biology* 24.4 (2019), pp. 811–821.

[153] Patrick Winter. "Deep Learning als Virtual-High-Throughput Screening-Methode unter Verwendung von gerasterten Molekülstrukturen". PhD thesis. Konstanz: University of Konstanz, Apr. 2020.

[154] Patrick Winter, Christian Borgelt, and Michael R Berthold. "Learned Feature Generation for Molecules". In: *International Symposium on Intelligent Data Analysis*. Springer. 2018, pp. 380–391.

[155] David Wolpert. "Stacked Generalization". In: *Neural Networks* 5 (2 1992), pp. 241–259.

[156] Lei Xu and Shun ichi Amari. "Combining Classifiers and Learning Mixture of Experts". In: *Encyclopedia of Artificial Intelligence*. Hershey, PA, USA: IGI Global, 2008, pp. 318–326.

[157] Yumeng Ye, Haichun Liu, et al. "Improvement of Resting-state EEG Analysis Process with Spectrum Weight-Voting based on LES". In: *arXiv preprint arXiv:1712.07369* (2017).

[158] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *arXiv preprint arXiv:1511.07122* (2015).

[159] Matthew D Zeiler. "ADADELTA: an adaptive learning rate method". In: *arXiv:1212.5701* (2012).

[160] Harry Zhang. "The optimality of naive Bayes". In: *AA* 1.2 (2004), p. 3.