Neuroscience-Inspired Analysis and Visualization of Deep Neural Networks

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieurin (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von M.Sc. Valerie Krug

geb. am 11.09.1991 in Burgstädt

Gutachterinnen/Gutachter

Prof. Dr. Sebastian Stober
Prof. Dr. Eirini Ntoutsi
Prof. Dr. Peter Knees

Promotionskolloquium am 22.01.2024

Magdeburg, den 12.02.2024

Otto-von-Guericke University Magdeburg

Department of Computer Science
Institute for Intelligent Cooperating Systems

PhD Thesis

Neuroscience-Inspired Analysis and Visualization of Deep Neural Networks

Author:
Valerie Krug

12.02.2024

Supervisor
Prof. Dr.-Ing Sebastian Stober

Department of Computer Science
Institute for Intelligent Cooperating Systems
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Krug, Valerie:
Neuroscience-Inspired Analysis and Visualization of Deep Neural Networks
PhD Thesis, Otto-von-Guericke University
Magdeburg, 2024.

Contents

Abstract

Deep Neural Networks (DNNs) are very successful in various fields of application. Their success, however, is mostly achieved by increasing the model complexity in terms of types of architectures or the number of neurons. At the same time, it becomes harder to interpret how DNNs solve their learned task, which can be risky in critical applications like autonomous driving or decision-making systems in healthcare. There are several techniques to elucidate inner workings of DNNs and to explain their decisions for individual examples. However, most of them are projecting information to the input space such that results are typically most useful for images. For many other types of data, for example, audio, visually inspecting the data is not intuitive such that the results of the explanation techniques are less intuitive to interpret, as well.

In this thesis, we aim to develop a more generally applicable method for getting insight into DNNs which does not depend on input data that is visually interpretable. Instead of projecting information to the input space, we implement a comparative analysis of DNN responses to different groups of inputs. For pre-defined groups that comprise related input examples, for example, from the same category, we obtain typical activity of the DNN for each of the groups. If the similarities of activity are in agreement with the human expectation about the similarities of the groups, it indicates that the model learned meaningful concepts about the data. For this comparative analysis, it is only necessary that a person can estimate relative similarities and differences between the groups, that means, telling whether two groups are more similar to each other than to another group. For example, a person might estimate that a spoken ‘a’ and ‘e’ are more similar to each other than to a spoken ‘t’ and expect this to be reflected in the similarities of DNN activations, too. In this thesis, we use audio, specifically speech in an Automatic Speech Recognition (ASR) task, as a representative data that is not visually interpretable.

To develop such technique, we take inspiration from neuroscience because real brains are even more complex and opaque systems than DNNs and have been studied for decades. Therefore, we research how well-established methods from neuroscience can be adapted to analyze and visualize DNNs.

In particular, we present two methods that build on top of each other. First, we introduce a Event-Related Potentials (ERPs)-inspired technique to characterize DNN activity for groups of examples, which we call Neuron Activation Profiles (NAPs). This approach allows to describe how the DNN responds to certain pre-defined groups of inputs and how the responses differ between these groups. For example, this can reveal which neurons are activated for specific groups and which activate for all similarly. While this technique allows for comparisons, the activations themselves are not directly interpretable. Therefore, as a second technique, we present a visualization of activations as topographic activation maps, similar to how brain activity is visualized in Electroencephalography (EEG).

Computing NAPs and visualizing them as topographic maps finally allows to compare group-characteristic DNN responses visually. This way, we provide a visual tool to inspect DNNs independent of whether the input data are visually interpretable themselves. We perform experiments with different image recognition models and for an ASR model which processes audio data of speech as a representative task with data that are not visually interpretable.

Zusammenfassung

Tiefe neuronale Netze (TNN) sind sehr erfolgreich in einer Vielzahl unterschiedlicher Anwendungsgebiete. Allerdings wird dieser Erfolg oft durch eine Erhöhung der Komplexität des Modells erreicht, beispielsweise indem kompliziertere Architekturen oder eine größere Anzahl von Neuronen verwendet werden. Gleichzeitig erschwert diese Komplexität, zu verstehen, wie ein TNN seine Aufgabe löst, was ein Risiko in kritischen Anwendungsgebieten wie dem autonomen Fahren oder Entscheidungssystemen in der Gesundheitsversorgung darstellt. Es existieren bereits verschiedene Methoden, um die inneren Prozesse von TNN nachzuvollziehen sowie deren Entscheidungen für einzelne Eingaben zu erklären. Die meisten dieser Methoden projizieren jedoch Informationen auf den Eingaberaum, weswegen deren Resultate am nützlichsten für Bilder sind. Andere Datentypen, beispielsweise Audiodaten, sind als eine visuelle Darstellung nicht intuitiv interpretierbar. Dementsprechend sind für diese Datentypen auch Erklärungsmethoden, die eine visuelle Erklärung im Eingaberaum erzeugen, nur schwer interpretierbar.

Das Ziel dieser Arbeit ist es, eine Methode zu entwickeln, die Einblick in innere Strukturen und Prozesse von TNN gewährt, dabei aber unabhängig davon ist, ob die Eingabedaten visuell interpretierbar sind. Anstatt Informationen auf den Eingaberaum zu projizieren, vergleichen wir, wie ein TNN auf unterschiedliche Gruppen von Eingaben reagiert. Dazu ermitteln wir die Aktivierungen des TNN für vordefinierte Gruppen, welche zusammengehörige Eingaben beinhalten, beispielsweise Eingaben der gleichen Kategorie. Wenn die Ähnlichkeiten der Aktivierungen mit der menschlichen Erwartung über die konzeptuellen Ähnlichkeiten der Gruppen übereinstimmen, deutet das darauf hin, dass die vom Modell gelernten Konzepte sinnvoll sind. Um diese vergleichende Analyse durchzuführen, muss eine Person nur die Ähnlichkeit der Gruppen einschätzen können, das bedeutet, ob zwei Gruppen ähnlicher zueinander sind als zu einer dritten Gruppe. Zum Beispiel kann eine Person einschätzen, dass ein gesprochenes ‘a’ und ‘e’ sich ähnlicher sind als ein gesprochenes ‘t’ und dementsprechend erwarten, dass diese Ähnlichkeit in den Aktivierungen des TNN widergespiegelt ist. In dieser Arbeit nutzen wir Audiodaten, speziell

Sprachaufnahmen zur Anwendung für die Spracherkennung, als repräsentative Daten, die nicht visuell interpretierbar sind.

Um eine geeignete Methode zu entwickeln, dienen uns die Neurowissenschaften als Inspiration. Im Vergleich zu TNN sind Gehirne noch komplexere und undurchsichtigere Systeme, die seit Jahrzehnten untersucht werden. Darum erforschen wir, wie etablierte Methoden aus den Neurowissenschaften auf TNN übertragen werden können, um diese zu analysieren und zu visualisieren.

Im Speziellen stellen wir zwei Methoden vor, die aufeinander aufbauen. Erstens zeigen wir einen Ansatz, um TNN Aktivierungen für Gruppen von Eingaben zu charakterisieren, welchen wir Neuronen-Aktivierungs-Profile (NAP) nennen. Diese Methode ermöglicht es, zu beschreiben, wie ein TNN auf bestimmten vordefinierte Gruppen von Eingaben reagiert und wie sich diese Reaktionen zwischen den Gruppen unterscheiden. NAP erlauben es, Gruppen miteinander zu vergleichen, allerdings sind die Aktivierungen, die sie beschreiben, nicht direkt interpretierbar. Aus diesem Grund stellen wir eine zweite Methode zur Visualisierung der gruppenspezifischen Aktivierungen als topographische Aktivierungskarten vor. Diese Visualisierungs-Technik ist davon inspiriert, wie Gehirn-Aktivität bei der Messung mittels Elektroenzephalographie (EEG) dargestellt wird.

NAPs und deren Visualisierung als topographische Karte ermöglichen es, Reaktionen des TNN für unterschiedliche Gruppen visuell miteinander zu vergleichen. Damit stellen wir ein Werkzeug zur Verfügung, um unabhängig davon, ob Eingaben visuell interpretierbar sind, Einblicke in TNN zu gewinnen. Unsere Experimente umfassen unterschiedliche Bilderkennungs-Modelle sowie ein Spracherkennungsmodell, das Audiodaten verarbeitet, als repräsentatives Beispiel für Daten, die nicht visuell interpretierbar sind.

1 Introduction

Artificial Neural Networks (ANNs) have become a very popular tool for solving challenging tasks across various fields of application. Increasing their performance is often achieved through increasing their depth, the number of neurons or by using more complex architectures [178, 53, 186]. At the same time, larger computational models become black-boxes, which are harder to interpret [194]. This complicates detecting erroneous behavior and understanding how they perform their task, which can be risky in critical applications like autonomous driving or decision-making systems in healthcare. Several introspection techniques have been proposed to obtain insight into ANNs [196, 162]. However, most of them are designed for certain applications or architectures. In particular, many introspection techniques focus on images because the features of images are easy to interpret visually. That means, it is easy for a human to identify parts of an image or patterns that are characteristic for a particular object. Therefore, humans can compare this expectation with introspection technique results that highlight prediction-relevant input parts or visualize patterns that the ANN uses to perform its task. Features in other data, for example in the audio domain, are more difficult to interpret visually, making established introspection techniques less suitable for understanding the model. Humans can intuitively listen to sounds, but visualization of the digital representation of audio data is not intuitive. Therefore, for example, interpreting an explanation that visualizes which frequencies an ANN uses for a prediction is difficult to visually interpret, too. In this work, we address the consequent need for methods to also understand ANNs that process data which are not visually interpretable, specifically focusing on audio data of speech.

While complex ANNs are a recent technical development, real brains have been studied in neuroscience for over 50 years. Brains and ANNs share that they are highly complex and opaque information processing system. Therefore, the rich experience from the field of neuroscience to analyze such complex and opaque systems can potentially help to understand ANNs better, too. In this thesis, we follow the idea of using well-established methods that are used for understanding real brain activity and adapting them to be used to get insight into ANNs.

1.1 Research Aims

Our aim is to develop a novel approach on performing Deep Neural Network (DNN) model analysis that does not require visually interpretable data but works by comparative analysis of the model's activations for different groups of inputs. For a set of predefined groups of inputs, for example, groups that belong to different categories, we want to provide a means to investigate similarities and differences in how the DNN activates for these categories. This allows a person to evaluate whether the representation of concepts in a DNN is reasonable based on the group similarity. For example, consider a model that classifies animals in an image. Intuitively, a human would expect that a cat and a lion are more similar to each other than to an elephant. If such similarities are reflected in the activations of a model, this indicates that the learned concepts are reasonable without the need of characterizing them explicitly. As another example, consider a speech recognition system that turns spoken sentences into text. We would expect similar sounds (like a spoken 'a' and 'e') to activate more similar to each other than to different sounds (like a spoken 't'). If this was not the case, it is doubtful for a person to believe that the speech recognition is based on actual acoustic concepts. The expected similarities can also be preferences about the model representations. For example, if we prefer an image recognition model to not distinguish people based on their skin color, we would trust the model more if the DNN activates similarly for any group of people of similar skin color. We like to emphasize that in this specific example, the model does not necessarily discriminate against any group of people with respect to its output only because internal representations differ between the groups.

To be able to perform such comparative analysis of DNN activity for different groups, we require two main components. First, we need a method to characterize how the DNN responds to groups of inputs. Second, as activations typically cannot directly be intuitively compared, we investigate how to visualize these responses to allow for more intuitive visual comparisons. These requirements motivate our following Research Aims (RAs) and to pursue them, we adapt methods from neuroscience.

RA1: Characterize group-specific network responses with minimal information loss from aggregation. For our first aim of characterizing how a DNN responds to groups of inputs, we take inspiration from a popular technique in the field of neuroscience, the Event-Related Potentials (ERPs). The ERP technique is used for analyzing brain activity through Electroencephalography (EEG) [112]. ERPs aim to measure brain activity for a particular fixed event (stimulus) and average over multiple measurements (trials) to denoise them. We analyze DNNs similarly, but as their responses are deterministic, we use the averaging to remove particular variations in the input data. For example, in a speech recognition model, we characterize DNN responses to a particular sound by using averaging to remove the variation that originates from different speakers and articulations. Like the fixed event in ERPs, a challenge of this averaging approach is that data and the ANN's activations need to be aligned at the feature of interest. In the DNN case, this means that the relevant patterns of interest in different inputs or activations need to be aligned before applying the averaging.

RA2: Summarize complex representations through visualization. To obtain a more intuitive way of comparing activity between groups, we aim to be able to visually compare it. Specifically, we aim to achieve this by adapting how brain activity is typically visualized as a topographic map. For example, brain activity recorded through EEG measurements [112] is represented as a top view of the head with a superimposed topographic map of neural activity [113]. Adapting this approach to be applicable to DNNs can help to visualize and understand their internal representations more intuitively, too. We would be able to see less and more active sets of neurons as regions and could visually compare these regions between different classes to identify commonalities and differences in the DNN activations. However, in contrast to the brain, DNNs typically do not have an informative order of neurons because there are no connections between neurons within the same layer. Therefore, to be able to visualize activations of neurons in DNNs as topographic maps, we research techniques to layout the neurons in a two-dimensional space in which neurons of similar activity are in the vicinity of each other. The idea of introducing a topographic neuron layout in ANNs is not novel. Self-Organizing Maps (SOMs) [81] follow a similar motivation and already constrain the neurons to form a topographical layout during training. However, traditional SOMs are only shallow neural networks and more recent approaches on training deep SOMs [102] have not gained popularity as the topography constraint can decrease performance. Most DNNs that are used in practice are implemented without a topographical layout of the neurons. Our aim is to provide the possibility to create a topographic layout visualization for any DNN, particularly those that are already trained and potentially deployed in the real world.

RA3: Provide analyses and visualizations that do not rely on visually interpretable data. This research aim cannot be addressed in isolation but accompanies RAs 1 and 2. We aim to provide analyses based on the ERP-inspired characterization of network responses and topographic activation maps that do not require visually interpretable data. We intend to achieve this by performing comparative analyses of DNN activities instead of projecting information to the input space. In this thesis, we use DNNs for audio processing, specifically Automatic Speech Recognition (ASR), as representative models that process data which are visually difficult to interpret.

1.2 Structure of the Thesis

This thesis comprises nine chapters. We start with a description of background and related work. Then, we introduce data and models we use in this thesis. This is followed by the three main chapters of the thesis which address the respective three research aims. Afterwards, we show promising preliminary work and extensions of our methodology. Finally, we conclude the thesis.

Chapter 2 presents the fundamentals of Machine Learning (ML) and Deep Learning (DL) techniques and Chapter 3 introduces existing approaches of analyzing and visualizing DNNs.

In Chapter 4, we introduce the different data sets and models that we use in the thesis. Furthermore, we provide an overview of where we use the different models in the thesis.

Chapter 5 and Chapter 6 are the central chapters of this thesis as they present and evaluate our proposed neuroscience-inspired approaches to pursue RAs 1 and 2, respectively. Both chapters focus on introducing the respective method and evaluating it with suitable experiments. Application examples are provided in the separate Chapter 7.

In Chapter 5, we introduce our novel ERP-like analysis called Neuron Activation Profiles (NAPs) which addresses RA1 of characterizing group-specific DNN responses. First, we provide a detailed description of the individual steps of computing NAPs. Then, we organize them in a computation pipeline with optional steps depending on the use case. For the critical steps in the pipeline, which are averaging, normalization and alignment, we perform a more detailed evaluation. To validate our introduced alignment procedure, we show toy examples based on MNIST and a real-world data set with expected alignment positions.

Creating easily interpretable visualizations of DNN representations for RA2 with topographic activation maps is presented in Chapter 6. This chapter builds on Chapter 5 as we visualize NAPs as topographic maps. Our aim for Chapter 6 is to find a method to layout neurons in a two-dimensional space in order to visualize them similar to a topographic activation map in neuroscience. To this end, we introduce and evaluate different layouting techniques that project the artificial neurons in a two-dimensional layout where similarly activated neurons are in the vicinity of each other. We perform this evaluation qualitatively by subjectively inspecting whether the results visually resemble their neuroscientific counterpart. Also, we investigate several measures to quantify the visual quality of the resulting visualization. For the best suitable layouting technique and visual quality measure, we further evaluate how alignment and aggregation techniques affect the visual quality and whether there is a difference between layers in the network.

Using NAPs and topographic activation maps, we discuss their applicability to different models and data sets in Chapter 7. This evaluation of their applicability includes addressing RA3 as it demonstrates that the analyses and visualizations are useful regardless of whether the input data are visually interpretable. This chapter begins with a discussion on how well the techniques scale with different model and data properties. Then, we show several examples that demonstrate the applicability of our technique to different models. Specifically, this involves a detailed analysis of the representations of an ASR model with NAPs to show that we do not depend on visually interpretable data. Moreover, we demonstrate how topographic activation maps can be used to detect errors and to show representational bias in DNNs.

In Chapter 8, we present extensions of the methodology that do not address the research aims but provide additional exemplary applications of our novel approaches. In particular, this involves a discussion on extending our visualization to multiple layers and visualizing training processes with topographic activation maps. Moreover, we show an experiment on the relation of DNN activations and its confidence in terms of output softmax value using our visualization technique. Finally, we discuss how to potentially use our visualization technique to use established explanation approaches in a more targeted way.

Finally, Chapter 9 concludes this thesis. We summarize the findings and discuss whether we were able to meet our research aims. Based on this, we discuss most promising future research directions.

1.3 Publications

Parts of this thesis have been reviewed and published. We will point out the references in the respective sections. In this section, we provide an overview of our relevant publications and how they are integrated in this thesis.

The largest contributions are published in "Analyzing and Visualizing Deep Neural Networks for Speech Recognition with Saliency-Adjusted Neuron Activation Profiles" [89] and "Visualizing Deep Neural Networks with Topographic Activation Maps" [92]. The first publication introduces our most advanced version of NAPs and their application to an ASR model. Different to this publication, we present NAPs as a more modular framework in Chapter 5 because not every step of the ASR-suitable pipeline is necessary for every application. From this paper, we report the speech-based alignment evaluation in Section 5.3 and the ASR application in Section 7.2. The topographic activation maps publication [92] builds the basis for Chapter 6 and the application examples of error and bias detection in Sections 7.3 and 7.4. Different to the publication, we introduce size-weighted convexity as an additional quality measure for topographic activation maps. We further evaluate the suitability of the measures using manually created topographic maps in Section 6.3.2 and focus on results using the most suitable measure in the main thesis in Section 6.3.3. Moreover, we extend the bias detection example in Section 7.4 to more sensitive variables, multiple layers and including the visual quality measure.

In earlier publications, we show initial versions of our NAP approach. With "Introspection for Convolutional Automatic Speech Recognition" [85], we introduce "Normalization and Averaging of Aligned Inputs (NAvAI)" which can be considered an input layer NAP with saliency alignment but without masking prediction-irrelevant positions. Therefore, plotted input layer NAPs in Section 7.2.2 are related to this publication. "Neuron Activation Profiles for Interpreting Convolutional Speech Recognition Models" [88] is our first extension of the averaging approach to activations. This work presents a version of NAPs that does not use alignment but tries to be time-independent by comparing the distribution of neuron activity in different feature maps. From this work, we keep the idea of applying the averaging approach to hidden layers but decide to use the saliency-based alignment to obtain better representative NAPs. Further, we use the same similarity-based clustering of NAPs and its visualization as heat maps with dendrograms (clustermaps) as reported in Section 7.2.2.

Recently, parts of this thesis were accepted for publication. "Visualizing Bias in Activations of Deep Neural Networks as Topographic Maps" [90] covers the aforementioned extension of the bias detection example in Section 7.4. "Relation of Activity and Confidence when Training Deep Neural Networks" [91] presents an experiment that uses topographic maps to investigate activity in dependence of the confidence of a DNNs, which is reported in Section 8.3.

In this thesis, we create topographic maps based on activation similarity. However, earlier, we investigated to create such topography in the filter space, following the assumption that similar filters lead to similar activations. Arranging the filter space in a topographic layout is included in "Adaptation of the Event-Related Potential Technique for Analyzing Artificial Neural Networks" [84], our first first version of ERP-based analysis of DNNs. More explicitly, we investigate the topographic layout of filters in "Visualizing Deep Neural Networks for Speech Recognition with Learned Topographic Filter Maps" [86]. Based on the unexpectedly low correlation of filter similarity and activation similarity, we decided to optimize for activation similarity directly when creating topographic activation maps. Therefore, results from these publications are not included in this thesis.

2 Background

In this chapter, we introduce relevant background knowledge for this thesis.

We begin with three sections on Machine Learning (ML)-related topics. Section 2.1 introduces general concepts of ML, including different learning types like supervised and unsupervised learning, as well as selected ML techniques. In particular, this involves clustering approaches, Artificial Neural Networks (ANNs), Self-Organizing Maps (SOMs) and Particle Swarm Optimizations (PSOs) because they are used in this thesis. As we focus on Deep Neural Networks (DNNs) in this thesis, Section 2.2 explains how to train and evaluate DL models and introduces various components for building them. Thirdly, we present common dimensionality reduction techniques in Section 2.3.

In this thesis, we use image and audio processing tasks with DNNs. As DL for audio data is not as commonly known as image processing, we introduce fundamentals of processing recorded audio for using it as input data in DL in Section 2.4.

Finally, because we use methods that are inspired by neuroscientific techniques, in particular from Electroencephalography (EEG) data analysis, we introduce fundamental techniques in Section 2.5. Specifically, we present the Event-Related Potentials (ERPs) technique and visualization of brain activity as topographic maps.

2.1 Machine Learning

ML comprises algorithms that are designed to perform tasks without explicitly programming them. To this end, ML algorithms learn from data and, in some cases, annotations about this data, to solve a defined task [43]. This is particularly useful for tasks that are too complicated to be implemented with manual instructions.

2.1.1 Supervised Learning

In supervised learning, the algorithm is trained on data for which the target output is known. To this end, the training requires providing the algorithm data in which each example is annotated with its expected output. The algorithm learns to map the input data to the correct output by minimizing the difference between its predictions and expected output. There is a wide range of typical tasks in supervised learning, for example classification, regression or transcription.

Classification is a learning task, where the target output is predicted from a set of possible categories. Such classification can be binary, where the model only distinguishes between two classes. For example, binary classification has applications in detection of diseases [119, 3] or to detect defects in industrial applications [26]. If there are more than two classes but only exactly one of them is the expected output, it is a multinomial classification task. This is common in applications like image classification [105, 163, 16] or speaker recognition [49]. Finally, multi-label classification is a task where there are multiple possible output categories, but each example can have more than one correct output prediction. Typically, this involves tasks where output categories are not excluding each other like in text sentiment classification where a statement can be, for example, both fearful and worrying at the same time [103]. Also, this applies when output categories follow a hierarchy, for example, in respiratory disease detection which simultaneously predicts the presence and the specific type of the disease [18].

Regression tasks are predicting continuous target values instead of categorical ones [99]. This is, for example, applied in weather forecasting [58], stock market predictions [165] or prediction of prices at the housing market [110].

Translation and transcription are tasks of predicting a sequence of outputs for a given sequence of inputs. If input and output data come from the same domain, for example, when translating words or sentences of one language to another language [174], the task is referred to as translation.

Transcription describes a translation from one domain to a different one. Typical transcription applications are Automatic Speech Recognition (ASR), which transcribes an audio input to a textual output [195] or phonemic transcription, which transcribes written language to a sequence of sound pattern descriptors, so-called phonemes [193].

In this thesis, we use models that are trained by supervised learning, particularly multinomial classification models for different image recognition tasks and transcription models for speech-to-text and speech-to-phoneme transcription.

2.1.2 Unsupervised Learning

Unsupervised learning involves ML models that learn from data without providing target outputs. This can either be necessary because the target outputs are not known or because the aim is to identify patterns, structures, or relationships in the data without specific guidance.

The unsupervisedly extracted information can also be used for consecutive tasks. For example, it facilitates anomaly detection [131] where the prediction is made by whether an observation differs too strongly from the usual patterns that the model learned for the data set. Anomaly detection is, for example, used to detect fraud in credit card transactions [8]. Moreover, extracted features through unsupervised learning can be used as inputs for a supervised classifier which needs fewer annotated training examples because the feature extraction does not need to (entirely) be learned together with the classification. This approach is referred to as unsupervised pre-training [34].

Common approaches for unsupervised learning include clustering techniques and ANNs. A recently less commonly used method, which we use in this thesis, are Self-Organizing Maps (SOMs). Some dimensionality reduction techniques, which we describe separately in Section 2.3, are unsupervised learning approaches, too.

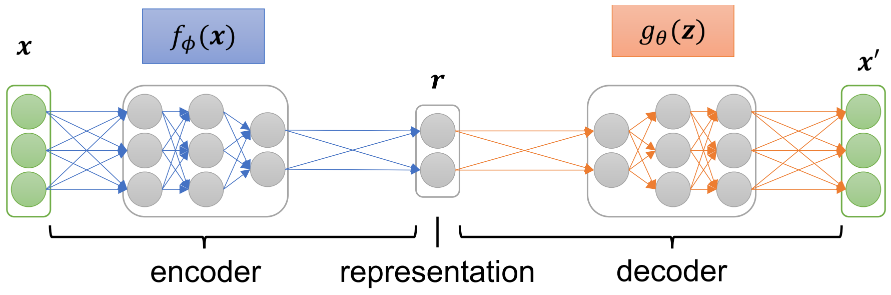
Clustering

Clustering algorithms group data according to their similarities [64]. Data can be optimally assigned to a selected number of clusters, for example, in k-means clustering [52]. Setting a fixed threshold for cluster count, however, can be inflexible. As an alternative, hierarchical clustering techniques describe a hierarchy of data examples based on their similarities [126]. Then, a threshold can be set flexibly to divide the data into clusters based on this hierarchy. A very commonly used clustering algorithm is DBSCAN [36], which detects highly similar sets of examples as core clusters and further groups points that are within a certain neighborhood size of this initial core. The quality of clustering is typically evaluated with measures that reward if distances within each cluster are smaller than between clusters, for example, the Silhouette score [155]. There also are fuzzy clustering algorithms [192] that do not assign a strict partitioning but probabilities of being in each cluster.

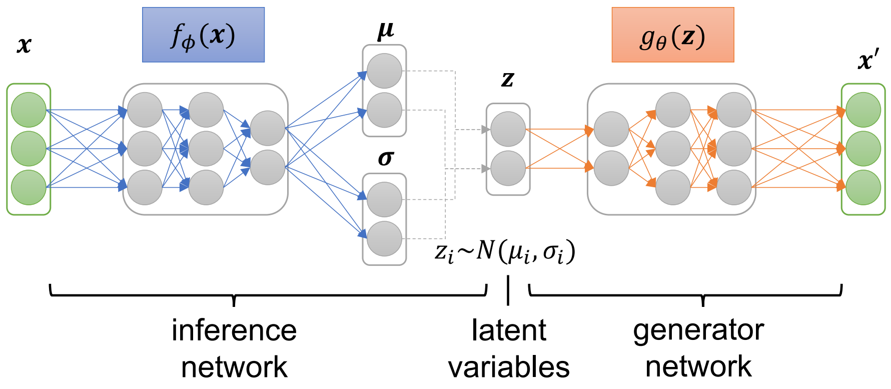
Artificial Neural Networks

ANNs for unsupervised learning follow different approaches. Early research employed energy-based models that modeled inputs and their learned features as random variables. The training of this kind of models is based on maximizing the likelihood of the data under the learned parameters [50]. That means, drawing a random sample from the trained model would yield a sample from the data distribution. Well-known models in this category are Boltzmann Machines (BMs), with the variations Restricted BMs, Deep BMs and Deep Belief Networks [96].

More recently, DNNs are more popular for unsupervised learning of data. This includes Autoencoder (AE) structures that are trained to reconstruct data while needing to compress the information in a low-dimensional representation. There are, for example, deterministic AEs [12] which compress information to a vector of numbers and probabilistic AEs like the variational AE [75] which learn to compress information as parameters of probability distributions. An illustration of the concept of these two AE variants is shown in Figure 2.1. A more comprehensive introduction to DNNs follows in Section 2.2.



(a) Deterministic Autoencoder. It encodes data as a vector of numbers.



(b) Variational Autoencoder. It encodes data as parameters of random distributions (here, a normal distribution). This allows to draw samples from the variational AE by only using the inference network.

Figure 2.1: Illustration of a deterministic and a variational Autoencoder. Green indicates input and output pairs which, in the case of autoencoding, are expected to be the same. Blue are network components that compress the input to a low-dimensional representation. Orange are the respective network parts that decompress this representation to the data in the original dimensionality.

Self-Organizing Maps

SOMs [81] are ML models that learn a set of vectors to represent the data. Each data example is assigned to the vector it is closest to. The important property of this model is that the learned vectors are connected in a grid to preserve the topological structure of the data. The training process, as illustrated in Figure 2.2, is the following. For a data example, the closest vector of the SOM is obtained, and commonly referred to as the winner. This vector is then updated to be closer to the data example. In addition, all other vectors are as well moved closer to this example, but to a smaller extent depending on how far on the grid the vector is from the winner. Iterating over all data examples (multiple times) distributes the vectors along the grid and is expected to also distribute according to the data density. This means, at the end of training, there are more vectors in regions with high density than in low-density regions.

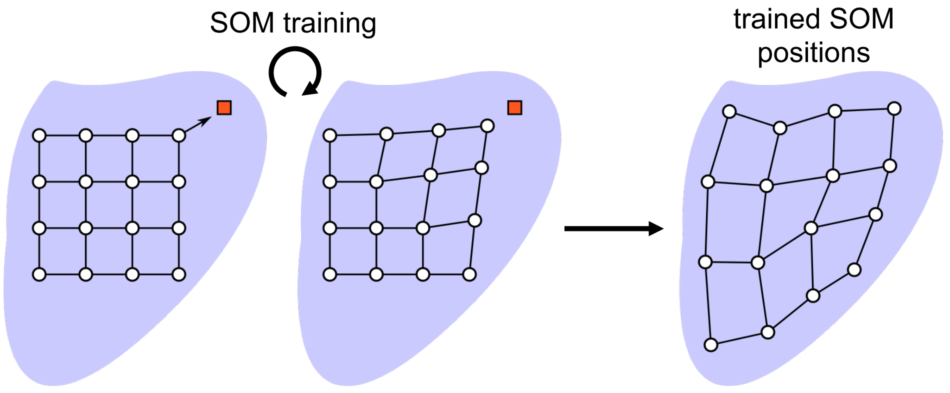


Figure 2.2: Illustration of a SOM training. Blue indicates the data distribution, white dots with connections are the SOM grid and the red square is the training example in one iteration of training.

2.1.3 Other Learning Types

Apart from supervised and unsupervised learning, there exist several other learning types. For example, semi-supervised and self-supervised learning are variations of the previously described approaches. Semi-supervised learning is a training method with a combination of data with and without target outputs [48]. Self-supervised learning is a version of supervised learning where no separate target outputs are given but (variations of) the data or the predictions by the model are used as output targets [35]. Reinforcement Learning (RL) is a conceptually different learning paradigm in ML, where a model, typically called RL agent, is given an environment to perform specified actions in as well as a reward function. Then, the RL agent explores actions in the environment and optimizes them to maximize the reward [7].

2.1.4 Particle Swarm Optimization

PSO [69, 166, 32] is a biologically inspired metaheuristic algorithm used to search for optimal solutions. It takes inspiration from swarm behavior of animals that head towards a common target and each individual is drawn towards other members of the swarm while also keeping some distance to each other. In the ML counterpart, we use a set of particles in the solution space and search the optimal solution by moving the particles based on simple mathematical formulas. Each individual particle follows simple local rules but it is influenced globally by all other particles allowing the entire swarm to collectively find an optimal solution. To increase the diversity in the swarm and avoid converging to local minima too fast, an attraction-repulsion PSO has been introduced [154]. In this variation of PSO, repelling particles are used to move particles out of potential local minima. Over optimization time, the number of repelling particles is typically decreased in order to converge at some point.

In this thesis, we use the idea of local and global influences on particles as well as the concept of attraction and repulsion forces. However, our usage is not a typical PSO because the particles do not aim to reach a common goal. Rather, we use the concepts of distribution to optimize a positioning of the particles. Details of our PSO-inspired techniques follow in Section 6.1.2.

2.2 Deep Learning

DL is a branch of ML that uses ANNs with multiple layers, hence called DNNs. DNNs are able to learn complex feature representations by learning simple patterns in the early layers and combining them to increasingly complex patterns in the deeper layers. In general, a DNN consists of many artificial neurons, each computing a weighted sum of its connected inputs and applying a non-linear function, the activation function, to the result. These neurons are organized in a series of hidden layers, where typically only neurons of successive layers are connected but not neurons of the same layer. Each of these connections is weighted by a trainable parameter. Because of the huge number of connections, resulting from having several hidden layers, DNNs need massive amounts of data as input to learn suitable connection weights for performing their task correctly. In Figure 2.3, a simple structure of a DNN is shown. More specifically, it shows an Multi-Layer Perceptron (MLP) with one input layer, two hidden layers with three neurons and one output layer.

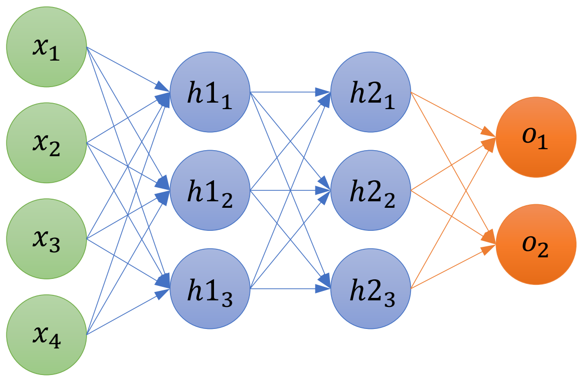


Figure 2.3: A simple Multi-Layer Perceptron. Green indicates the input layer, blue the two hidden layers and orange the output layer and their respective associated connections.

DL models are not restricted to this particular type of architecture. In the following sections, we explain how to train and evaluate the models in different learning tasks. Further, we present common building blocks of designing the layers and their connections. Finally, we explain transfer learning as a common training strategy to make use of unlabeled data in supervised tasks or to re-use layers of trained models for learning a different task. We give a explanatory overview of important concepts to follow this thesis. For further details about Deep Learning, we recommend available text books and surveys [45, 22, 27].

2.2.1 Training

DNNs are trained by defining their architecture, providing a data set and a function to optimize. This optimization target is also referred to as the loss function. The choice of the output layer of the network and the loss function depend on the task that the DNN is supposed to solve.

Supervised Learning

In supervised learning, target predictions for each example of the training data are available.

For a binary classification, the network requires one output neuron and for multi-label classification it needs a output neuron for each possible class.

Further, the output layer applies a sigmoid activation (Equation (2.1)) to each output neuron to obtain a probability value of being in each respective category.

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

The loss function then measures the discrepancy between the predicted output probabilities and the target classes that are provided as vectors with value 1 for each class the example belongs to and 0 for all others. For binary and multi-label classification, binary cross-entropy (summed over the neurons) is a suitable loss function. In a multinomial classification, the output layer of the DNN is expected to only predict high probability for one class. To achieve this, instead of using sigmoid activation, a softmax activation is applied over all output layer neurons. Softmax is the exponential value of the output layer activations, normalized by dividing by the sum of exponential values of all output neurons (Equation (2.2)).

$$\text{softmax}(\mathbf{o})_i = \frac{e^{\mathbf{o}_i}}{\sum\limits_{o \in \mathbf{o}} e^{o}} \text{ for each } i = 1, \ldots, |\mathbf{o}| \tag{2.2}$$

This way, the sum of post-softmax activations in the output layer is 1. Similar to the other classification settings, cross-entropy is a suitable loss function. As we only have one target output class, this can be computed efficiently by only considering the cross-entropy for the desired output class. This efficient version is commonly referred to as categorical cross-entropy.

Regression tasks often have no restriction on the value range of the output. Therefore, there is typically no activation function applied to the output layer. To compute a loss for optimization, the predictions can be compared with the targets with metrics like Mean Squared Error (MSE).

Unsupervised Learning

In unsupervised learning, there are no annotated target predictions for examples in the training data. Instead, a common strategy is to use the training examples themselves as a target. The DNN is built as an architecture where the input and output have the same dimensions. Then, the loss function is the difference between the original example and the example after being processed by the network, for example, measured by MSE. This process only yields useful models, if the network cannot copy information from the input to the output. This can, for example, be avoided by architectural choices, like the probabilistic layer in a variational Autoencoder. Alternatively, the input can be lightly perturbed, for example by adding noise, to prevent the possibility of copying information [187].

Optimization

Given the architecture, data and loss, the network weights can be optimized to minimize the loss function across the training data set. In DL, we aim to optimize the connection weights between neurons for this purpose. To determine, how to change weights to minimize loss, the backpropagation algorithm is applied [156]. Backpropagation computes the derivative of the loss function value with respect to each weight. Due to the layer-wise structure of DNNs, this involves a backwards layer-wise propagation of the derivatives. The backpropagation algorithm offers an efficient and largely paralellizable implementation of computing these derivatives. With the term 'gradients', we describe the collection of obtained derivatives of multiple (or all) weights.

Based on these gradients, the weights are updated. The simplest optimizer, Gradient Descent, changes the weights by adding the negative gradient. As the change would be large, this update it typically scaled down by a factor, referred to as learning rate. This factor can be used as a constant value or follow a schedule that, for example, adapts the learning rate over time or considers gradients from previous training steps, called momentum. Popular optimizers that use advanced learning rate schedules are Adam [74], AdamW [104] or RMSProp [54].

Large Training Data Sets

In DL, we use large training data sets for the models to be able to learn. This makes performing a step of optimizing weights for the entire data set computationally infeasible. Therefore, each training step is commonly performed for only small sets of data examples in a so-called mini-batch training. To this end, the training data is partitioned into small subsets, called mini-batches. Then, training is performed by iterating over the mini-batches and updating weights based on only the small set of examples. One iteration over all mini-batches is referred to as an epoch, that means, each epoch, the entire data set is seen by the model once. In mini-batch training, gradients in individual steps are only an approximation of the gradients of the entire data set. Therefore, while minimizing the loss for the current mini-batch, the overall loss might increase. In practice, by choosing small learning rates, individual small updates still lead to a successful optimization. Usually, it requires several epochs to training a model to minimize the loss sufficiently. To avoid seeing the same order of examples in every epoch, it is common to randomize the order of examples in every epoch.

Regularization

DNNs minimize the loss function for the training data. If the model is complex enough, it is therefore able to memorize the training data examples and the corresponding outputs. However, the actual aim of DL is to train a model which performs well on both training examples and unseen data. This property is also called being able to generalize to unknown examples. To achieve well-generalized models, regularization techniques make it more difficult for DNNs to only memorize examples. For example, common regularizers penalize highly-specific weights by adding a norm of the weights to the loss function. The sparsity regularizer L_1 penalizes the sum of absolute weight values which encourages learning as few weights as possible for solving the task. Weight decay (L_2) penalizes the sum of squared weight values prevents large weight values which the model might use as indicators for a specific example. Given weights θ, the regularizers added to the loss function are

$$L_1 = \sum_{i=1}^{n} |\theta_i| \qquad \text{and} \qquad L_2 = \sum_{i=1}^{n} \theta_i^2 \quad . \tag{2.3}$$

Regularization can also be achieved by perturbing the training data with noise or data-dependent alterations. For example, images can be rotated, sheared, cropped or resized and audio examples can be changed in speed or pitch. These perturbations need to be applied carefully to still retain realistic inputs for the given target.

Perturbations can also be applied to hidden layers of the network, again, for example, by adding noise to activations. Dropout is a related techniques that deactivates random neurons by setting their activations to 0. These approaches prevent that the model can rely on specific parts of the network or exact values of activations.

Finally, memorizing the training examples typically only occurs when training for many epochs because the DNN is trained on the same examples multiple times. Stopping the training earlier in training is therefore a common approach for regularization and is correspondingly called early-stopping.

Loss Function Extensions

Loss functions are not limited to the aforementioned cross-entropy and MSE. Any differentiable function that compares the output of the DNN to a target is usable. When using unbalanced data sets in classification, underrepresented classes can suffer from poor performance using regular cross-entropy. To counteract this, it is possible to weight the loss function such that prediction errors are higher penalized for rare classes. Further, loss functions often comprise different components, for example, one loss that penalizes prediction errors and one regularization term. Such multi-component losses are useful to optimize multiple targets simultaneously but need careful balancing of the strength of each component's influence.

2.2.2 Evaluation

If a model is not able to learn the training data, we speak of underfitting. However, often also the opposite happens and DNN memorize training data, which is referred to as overfitting.

Detecting underfitting is straight-forward from observing that the loss on the training set is not decreasing enough. Avoiding underfitting typically involves changing the model architecture, improving the loss function or testing for errors in the training data.

Overfitting, on the other hand, can only be detected if there is a data set available which the model has not been trained on. Therefore, it is unusual to train DNNs on all available data. Instead, we split the data set into three sets. The first is a large training set, which is used to train the model.

Second, we use a smaller validation set for optimizing hyperparameters like architecture, learning rate and regularization. Both of these data set parts are directly or indirectly used to optimize the model and hence not suitable for evaluation on unseen data. Therefore, a third small set of data is kept to only use it for evaluating the final model performance. All three data sets need to be disjunct and optimally they should not be from the same distribution. For example, if there is much redundancy in a data set, creating a random split of training, validation, and test data is not suitable to obtain independent sets of examples. In this case, it is advised to choose the data split based on knowledge about the data. For example, taking the training, validation and test examples from different conditions.

Using the data split, evaluation during training is performed with the training and validation data. Training data performance indicates whether the model learns well enough. The validation data set indicates whether it overfits on the training data. A learning process in which the training performance increases while the performance on the validation data stagnates or even decreases indicates overfitting. Based on the relation of the performance on the two sets, the model can be improved, for example, by stronger regularization in the case of overfitting. Finally, if the model is optimized like this, the test set performance is reported as measure of how well it generalizes to unseen data. Figure 2.4 shows idealized training and validation accuracy curves in the case of underfitting (low accuracy), overfitting (only high training accuracy) and a well-generalized model (both validation and training accuracy are high).

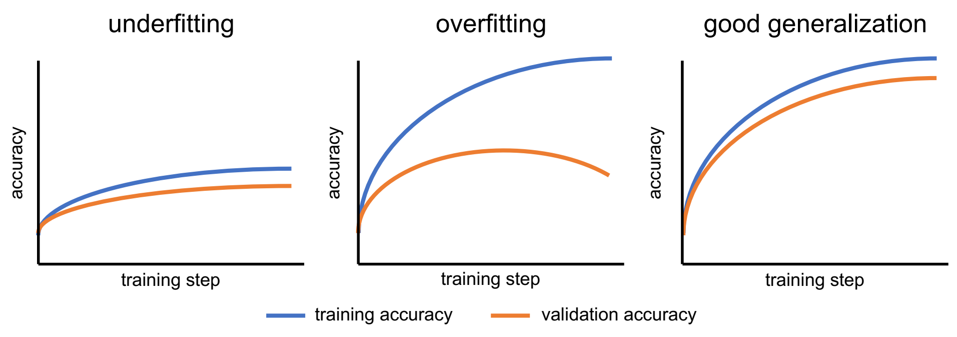


Figure 2.4: Illustrations of training and validation accuracy curves when a model is underfitting, overfitting or learning to generalize well.

As already indicated in Figure 2.4, evaluation metrics do not need to be the loss function. On the contrary, often we use a loss function because we need a differentiable target value but are interested in another performance metric. Most commonly, in classification, we are interested in measures of accuracy, precision and recall. As they are not differentiable, we train the model with cross-entropy but measure the final performance during evaluation with the actual target metric. Further, it is common to not only report global evaluation metrics like the accuracy over all test data examples, but also investigating performance on data subsets. For example, the performance for individual predicted classes can be reported and confusion matrices can indicate which classification errors are frequently made by the model.

Typical model evaluations focus on metrics, either directly via the loss function value or non-differentiable evaluation metrics like accuracy. This evaluation gives information that a model performs the task but does not reveal how the model solves the task. This shortcoming of evaluation is tackled by model introspection techniques that are described in Chapter 3 and which is also a focus of this thesis.

2.2.3 Architectures

DNNs can be built from a variety of components. We will explain the most commonly used architecture components in the following paragraphs.

MLPs

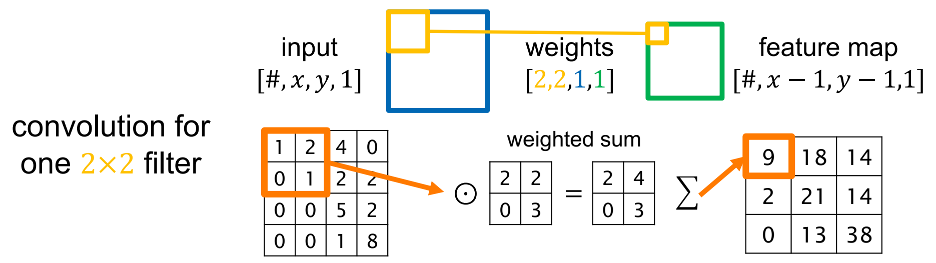
The simplest DNN architectures are Multi-Layer Perceptrons (MLPs) which are composed of several fully-connected layers (as shown in Figure 2.3). As the name suggests, fully-connected layers connect every neuron of a layer with every neuron in the succeeding layer. MLPs have, in theory, a high capacity to learn tasks due to their large number of trainable parameters. However, in practice, the same property complicates the learning process for larger models or difficult learning tasks. Therefore, MLPs are rarely used for practical real-world applications. Nevertheless, fully-connected layers are an important building block in many architectures. For example, the output layer of classification model requires a fixed amount of neurons which is commonly implemented by a fully-connected output layer.

CNNs

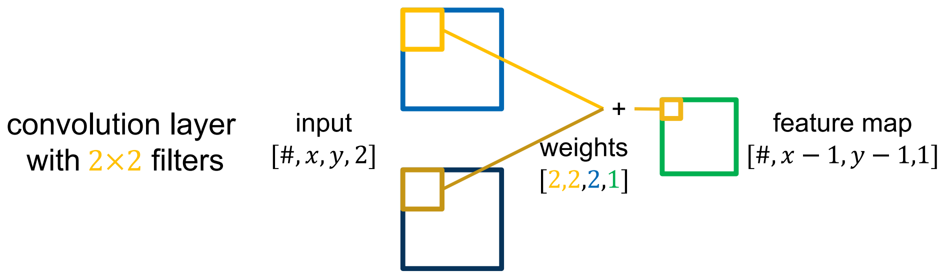
MLPs are wasteful in terms of parameters and, additionally, for detecting a particular pattern in different positions of the input, they need to learn it for each position separately. Convolutional Neural Networks (CNNs), in contrast, can detect patterns independent of their position in the input by convolving the input with trainable filters. This means, instead of applying different weights for each position, they learn small sets of weights that are applied across the entire input. The receptive field of a filter describes the region in the input that is used to compute a single output value of the convolution layer. The receptive field of filters in deeper layers are composed of the receptive fields of the connected positions in previous layers. Correspondingly, the receptive field of the output layer is the part of the input that is used for one prediction, up to the entire input in classification tasks. An illustration of the convolution operation for one filter and in a DNN setting where layers comprise multiple channels to convolve over with individual filters are shown in Figure 2.5.

Convolutions are typically applied to consecutive regions in the input. For reducing the resolution of the output, convolution can be applied with a stride of s, meaning that it is applied to every s^{th} region. To increase the receptive field without adding layers or increasing filter size, dilated convolutions can be used. These filters are not applied to adjacent positions in the input but only to every d^{th} position, depending on the dilation rate d. This means, the receptive field is larger but not every position in this larger receptive field is processed in one convolution operation.

The output of applying one convolution filter to the complete input is called a feature map [45]. Each position in a feature map is a neuron of the CNN, but due to the convolutional architecture not every pair of neurons in subsequent layers are connected. This sparse connectivity and weight sharing allows CNNs to perform tasks with substantially fewer parameters than MLPs with fully-connected layers. CNNs are particularly suitable for data in which features are spatially or temporally related, for example, images or sequential data. Depending on the dimensionality of the data, convolution filters can be used in different dimensionality as well. For example, images are commonly processed with two-dimensional filters and one-dimensional filters are better suitable for sensor signal processing.



(a) The convolution operation visualized for one filter. The output in one position is computed as a weighted sum of a window in the input with the convolutional filter, as highlighted in orange for one exemplary window.



(b) A convolutional layer with two input channels and one output channel. For each input channel, a convolution operation like in Figure 2.5a is performed and the sum over these results is used as a output in the output channel. There is a learned filter for every input-output-channel pair.

Figure 2.5: Illustration of convolution. Blue shades indicate input channels and green indicate output channels. Yellow indicates the input and outputs areas affected by one convolution. For one operation, an area of the input channel of the same size as the convolutional filter is affected and produces a single output value.

The output of a convolution is smaller than the input because for each applied window, only a single value is computed. In DNNs, it is sometimes desired to not change the dimensionality when using a convolutional layer. Therefore, convolutions can be performed with different padding modes. This means, before applying the operation, the input is enlarged by appending values (typically zeros) at the beginning and end of the input dimensions. Valid padding mode refers to not using padding and has the advantage that only values from the input data are used in the convolution. Same padding mode increases the input size such that the output size is equal to the size of the original image. It is typically used if the size of the feature maps has to remain constant, for example, if they are supposed to be concatenated. Finally, full padding aims to guarantee that every input position is visited by the convolutional filter the same number of times. However, in practice, it is uncommon to use full padding as the gained information is rarely improving the performance of models.

RNNs

While CNNs can process sequential data, it is difficult to process sequences of variable length. Moreover, they are limited to their receptive field to make a prediction. A neural network type that is able to process sequences of a variable length while using information from the entire sequence to produce an output are Recurrrent Neural Networks (RNNs).

RNNs are characterized by a recurrent connection in their hidden or output layer. This means that there is a regular neural network structure for each element in the sequence, which applies the same weights to every element. In addition, there are recurrent connections of layers such that

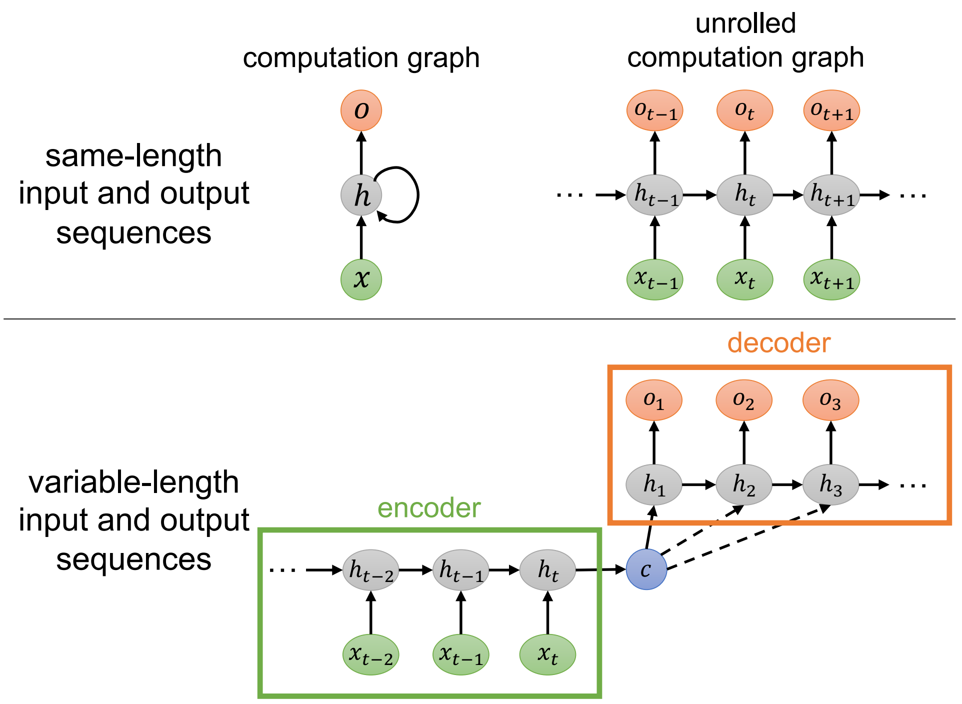


Figure 2.6: RNN architectures with one hidden layer h (gray) visualized for tasks where input x (green) and output o (orange) sequences are of same length and of different lengths. Variable-length requires an encoder that outputs a single context vector c (blue), which is used as input for a decoder.

the representation of one sequence element influences the succeeding element (or the other way round). Figure 2.6 shows conceptual illustrations of RNN architectures. RNNs that process sequences where the output has the same length as the input can predict one output element per input element. For input and output sequences of different length, the input sequence is encoded by a RNN into a single context vector (encoder) which is then used as an input to a second RNN (decoder) that predicts the output sequence. Either the context vector is provided to the first step of the decoder only (solid arrow from c in Figure 2.6) or as a constant input to every decoding step (dashed arrows from c in Figure 2.6).

Despite their architectural capability of accessing information from the entire sequence, hidden layers need to simultaneously perform a prediction task and the sequence memory task which is difficult to optimize. There are RNNs-specific building blocks that circumvent this problem by separating these two tasks into hidden state connections and a separate cell state that is carrying information about the sequence. For example, this involves Long Short-Term Memory (LSTM) cells [55] and Gated Recurrent Units (GRUs) [21].

Transformer

More recently, a architecture called Transformer has gained huge popularity [186, 30, 31, 14]. Like RNNs, Transformer models are suitable for processing sequences of variable length. However, instead of recurrent connection, they use attention mechanisms which allow the model to specifically learn which elements of the sequence to use information from. In particular, they use a so-called (multi-head) self-attention mechanism where the input data itself is used to determine which part of the input to use for further processing. This allows the model to learn complex relations between the elements in the sequence and use it to make predictions.

Other Building Blocks

In addition to layers, the choice of applied activation function can be considered an architectural choice, as well. While some layers require particular activation functions to work correctly, for example, output layers in classification that need sigmoid or softmax activation, other layers can generally use any (or no) activation function. There is a wide range of activation functions of which very popular ones are the Rectified Linear Unit (ReLU) and similar functions like leakyReLU or eLU, or sigmoidal functions like sigmoid and tanh [152].

There are also layers that apply functions without trainable parameters. The most common example are pooling layers, which are typically used in CNNs to reduce the dimensionality of feature maps and to introduce some translation invariance. Pooling layers apply an aggregation function like averaging or obtaining the maximum value in a small sliding window across the input, similar to convolution, as illustrated in Figure 2.7. Pooling can also be applied globally to aggregate feature maps to a single value for further processing, for example, to be able to use CNN models with inputs of variable size.

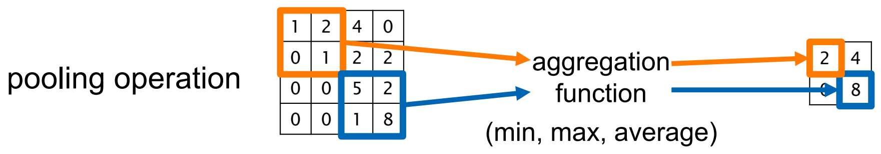


Figure 2.7: The pooling operation applies a non-trainable transformation to feature maps in a CNN. Different aggregation functions can be applied. The shown example uses maximum pooling with size 2×2 and stride 2. Blue and orange indicate different exemplary input windows where the maximum pooling is applied.

Connecting Building Blocks

The described building blocks of DNNs can be connected to each other flexibly [179, 53, 62]. Different to the classical layer-wise structure, layers that are not subsequent can also be connected by skip connections or residual connections. Multiple layers can also receive the same input to create architectures that process data in different ways and later combine them by, for example, addition or concatenation. The only important criterion for purely backpropagation-based training is that the entire architecture needs to represent a differentiable function.

2.2.4 Transfer Learning

Training DNNs from scratch can be challenging when compute resources are scarce or only few data are available. One common learning strategy in such low-resource tasks is to make use of an existing pre-trained model and adapting it to solve another task. This process is referred to as transfer learning. The underlying idea of transfer learning is that similar data share basic features but are differently combined in their specific context. For example, while speech from different languages is different, there are sounds that are shared in both languages. Therefore, features extracted by a pre-trained model of one task can be re-used for a similar related task.

Very popularly, this is used for image recognition tasks. Trained on the huge ImageNet data set which contains millions of images of thousand different categories, several high-performing models are openly available. These models can be used, excluding their top classification layer(s) as a feature extractor. The outputs of this stack of layers can be used as input for a smaller classification network to learn another image recognition task. This way, only few layers need to be trained and correspondingly fewer data examples are necessary for good model performance.

Often, the feature extractor weights are used without changing it during the transfer learning process. Setting these layers to have non-trainable weights is commonly called "freezing" the layers. However, it is also possible allow training of these layers and fine-tune the entire model. To prevent that the pre-trained layers unlearn information in the beginning of the transfer learning, it is common to first only train the classification layers with frozen feature extractor layers and then start the whole-model fine-tuning.

Unsupervised pre-training (compare Section 2.1.2) can be considered a type of transfer learning too. Instead of using layers of a pre-trained classifier to extract features, the feature extraction layers are used from a network that is trained unsupervisedly.

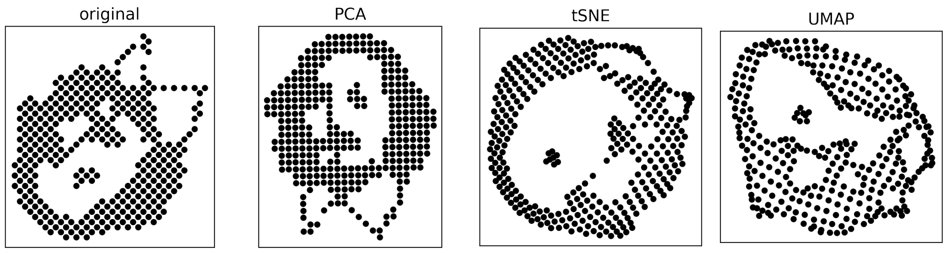
2.3 Dimensionality Reduction

Dimensionality reduction techniques project the high-dimensional data into a lower-dimensional space while preserving most information. The low-dimensional projection can be used for more efficient analyses or for visualizing data which has too many dimensions to visually inspect. Commonly used dimensionality reduction techniques are Principal Component Analysis (PCA) [60], t-Distributed Stochastic Neighbor Embedding (tSNE) [185] and Uniform Manifold Approximation and Projection (UMAP) [116]. Figure 2.8 shows exemplary results of the three methods when reducing differently complex data to two dimensions.

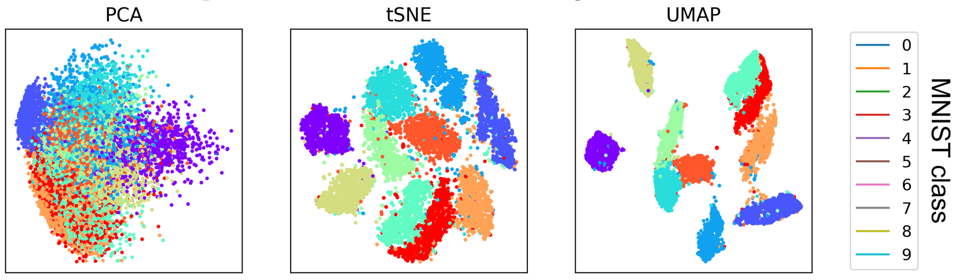
Principal Component Analysis (PCA) [146, 60, 66] is a traditional unsupervised technique for dimensionality reduction. The method linearly transforms the data points into a new coordinate system in which the new coordinates are the principal components, which are typically obtained from a Singular Value Decomposition (SVD) of the data matrix. The components are ordered by how much variance they explain in the data, which refers to how well the data distributes along the axis in the new coordinate system. Therefore, commonly, the first and second principal component are used for projecting the data into two dimensions because these explain most variance and therefore distribute the data maximally. However, relevant information can get lost if it is not contributing enough to the overall variation in the data. Moreover, because of the limitation to linear transformations, PCA is not able to project non-linear relations to the two-dimensional space.

tSNE was first introduced by van der Maaten and Hinton [185]. Like PCA, tSNE is an unsupervised algorithm but the projection is non-linear. tSNE optimizes the pairwise similarities in the low-dimensional space to be similar to those in the original high-dimensional data using a cost function. To this end, it constructs probability distributions over pairs of high- and low-dimensional data points, and minimizes the divergence between them. The initial coordinates for tSNE can be randomly chosen or, for more stability, can be initialized with coordinates from PCA. tSNE is particularly effective at preserving local structures. However, the global structures are not preserved as well such that clusters of data can appear which are not as distinct in the original data.

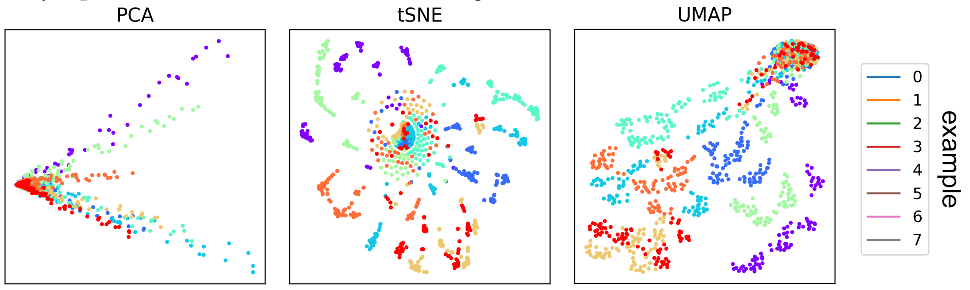
UMAP [116] is a recent non-linear dimensionality reduction algorithm. Different to tSNE, UMAP is based on a mathematical framework called Riemannian geometry. The algorithm assumes that the data points are uniformly distributed in a Riemannian manifold or uniformly distributed and the Riemannian metric is locally constant. It aims to approximate



(a) Applying dimensionality reduction techniques to an already two-dimensional input, more specifically, a picture of Otto von Guericke converted to a scatter plot, rotated by 225° clockwise. PCA is a linear transformation and, in the 2D case, only rotates the input. tSNE and UMAP reproduce local similarities but the global structure is distorted.



(b) Reducing the dimensions of the MNIST test data. Each point represents a MNIST example and is colored by the annotated class label. The dimensionality reduction is applied to the flattened pixel values of the examples. As the data is high-dimensional (784 dimensions), the non-linear techniques tSNE and UMAP are more suitable to find clusters than PCA. However, as indicated in the two-dimensional example in Figure 2.8a, they represent local structures better than global relations.



(c) DNN activations in a CNN trained on MNIST reduced to two dimensions. There are 8 dots for each feature map, activated by one out of 8 random input examples. The dots are colored by the respective example. All three techniques identify clusters of activations by the same example. In the PCA visualization they appear as linear clusters. tSNE and UMAP show multiple small example-specific clusters and one large dense mixed cluster. The focus on local similarities leads to a better distribution of points for tSNE and UMAP, but does not necessarily represent global structures well.

Figure 2.8: Results of performing dimensionality reduction for different inputs.

the topological structure of the data by constructing a low-dimensional representation that preserves both local and global structures, which is why the authors claim that UMAP preserves the global structure better than tSNE. However, there is also counter-evidence by Kobak and Linderman [80] who showed that, given the same initialization, UMAP does not perform substantially better than tSNE. Moreover, UMAP is preferred over tSNE for very large data set as it is computationally more efficient.

Both tSNE and UMAP have several parameters that can heavily influence the results. tSNE can be varied in perplexity, learning rate and the number of iterations. Parameters of UMAP are the number of neighbors and the minimum distance of points in the low-dimensional representation. Because of this dependence on parameter choices, it is recommended to not rely on findings from the two-dimensional representations but to validate them in the high-dimensional space.

Comparing the exemplary results in Figure 2.8, we see that tSNE and UMAP show better distributed and more clustered structures than PCA. This allows to interpret local similarities. Drawing conclusions about global relationships should be avoided, though. This is particularly visible in the two-dimensional examples in Figure 2.8a. Although there is a perfect representation of the input in two dimensions, tSNE and UMAP only recover local similarities, for example, the eye, nose, hair and tie, but distort the global structure such that the shape of the head is barely recognizable. We also observed tSNE and UMAP results in which the points belonging to the eye occurred outside of the head. On the other hand, in high-dimensional data, global structures are more complex and difficult to represent in two dimensions. Therefore, as visible from Figures 2.8b and 2.8c, PCA does not reveal clear clusters of related MNIST classes or input examples, respectively, in contrast to tSNE and UMAP.

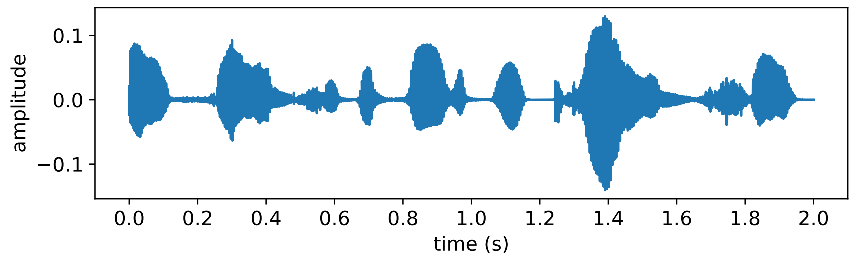
In summary, to project high-dimensional data to two dimensions, tSNE and UMAP are the most suitable methods. However, it is important to not rely on global relations between the data points. We consider the two techniques to be similar in terms of the quality of the results, particularly because the choice of parameters strongly affects the algorithms' outcomes. From this perspective, UMAP can be considered favorable compared to tSNE because it is computationally more efficient. This makes it better applicable the higher the dimensionality of the data is and, for individual results, it allows to optimize parameters over multiple runs in the same time as computing one tSNE projection.

2.4 Audio Processing

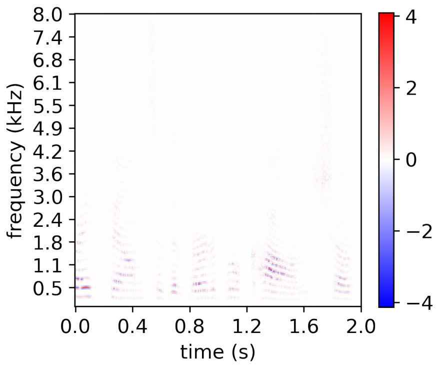
Processing audio data with ML model typically requires preprocessing steps. For recording audio, a microphone detects vibrations in the air. Their amplitudes are digitally stored and represent the raw audio data as a so-called waveform (Figure 2.9a). The vibrations have a high frequency, therefore, a large number of amplitudes per second needs to be stored to properly represent the acoustic input. This number is described by the sampling rate. Figure 2.9a shows a recording with originally 22050 Hz sampling rate which was downsampled to 16000 Hz. This means, in the shown time frame of 2 s, a sequence of 32000 amplitude values is visible. To make it easier for an ML system to process this data, it is common to convert the audio waveform into a more suitable format.

In particular, the spectrogram representation is very popular for processing audio with ML models [125]. A spectrogram is a representation that describes which frequencies are present in the audio signal over time. To create a spectrogram, a mathematical operation called Fourier transform is applied, which, for a given segment of audio, detects which frequencies it is composed of [136]. For obtaining information about how the frequencies change over time, the Fourier transform is not applied to the entire sequence but to small overlapping time intervals of the sequence. The length of these intervals is defined by the window size parameter and the overlap can be controlled by setting the hop size as temporal distance between two consecutive windows. This window-wise application is also called Short-time Fourier transform (STFT) [161]. From this approach, we obtain information about which frequencies are present over training time, that is, the spectrogram representation shown Figure 2.9b.

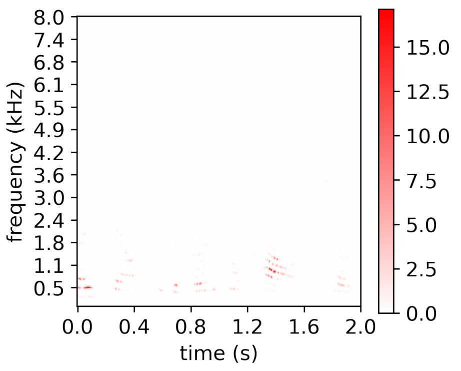
This spectrogram shows the amplitude of the respective frequency as positive and negative values but the sign of the value is not important. Therefore, the power is computed as the square of the values to obtain a power spectrogram (Figure 2.9c).



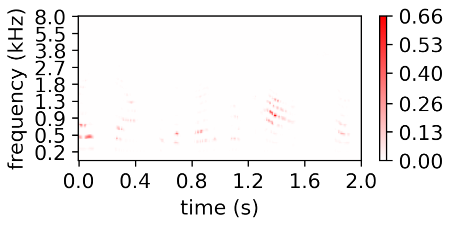
(a) Waveform.



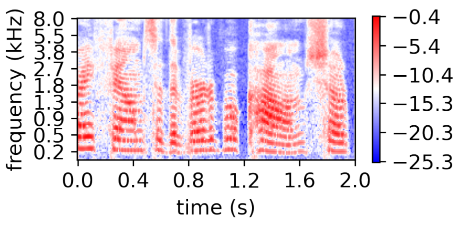
(b) Spectrogram.



(c) Power spectrogram.



(d) Mel spectrogram.



(e) Log mel spectrogram.

Figure 2.9: Overview of different audio data formats.

The linear frequency scale in regular spectrograms has the disadvantage that signals from natural speech are overrepresented in the lower frequency ranges, as visible in Figure 2.9c. Therefore, it is useful to adjust the frequency scale such that lower frequencies are considered in a higher resolution than higher ones. Using knowledge about natural speech, this can be achieved by using the mel scale [147]. By converting a the power spectrogram into a mel spectrogram, we see that the relevant information is now better emphasized (see mel spectrogram in Figure 2.9d).

Still, the power values in the mel spectrogram are typically concentrated at lower power values. This means there are many small values and only few high values. To improve the contrast in these low power regions of the scale, a logarithm is commonly applied to the mel spectrogram, leading to a log mel spectrogram shown in Figure 2.9e. In this final log mel spectrogram, we can nicely see log power values across the entire time and mel frequency range. This more structured format of a log mel spectrogram is finally used as input to the ML model.

It is noteworthy that converting the audio to a spectrogram is not lossless. Firstly, the Fourier transform results in complex-valued outputs whose imaginary part is typically neglected. Secondly, several processing steps are not invertible, like computing the squared value of amplitudes or binning frequencies when converting to mel frequencies. In practice, spectrograms are very informative about the original input. However, they are less suitable when there is a need to recover information from the original audio.

2.4.1 Speech Annotations

Speech data are audio recordings of human speech. There are a wide range of applications for processing speech with ML, like speaker recognition [49], voice assistant systems [61] or ASR [195]. Here, we particularly focus on ASR which is a transcription task from natural speech to a textual representation of the spoken content of the speech.

Intuitively, ASR is used for dictation tasks to obtain a written text of what a speaker says in the recording. In natural speech, it is impossible to exactly associate every individual letter to the sound within the spoken word because multiple letters can compose a single sound. Consider the word 'though', in which the letters 'ough' are pronounced as one sound. Moreover, such mapping is difficult to obtain because the same letter combinations can be pronounced differently, consider for example, the letters 'ough' in the words 'though', 'tough' and 'through'. Therefore, a data annotation of individual letters to specific time steps in a recording is not possible. Annotations of the spoken words can be provided for each word, that means, as information about which word is spoken in which time frame. More commonly, an entire sentence or text is provided for a longer recording of a person who reads this text. This missing alignment from letters to the input data is a challenge for supervised ML training.

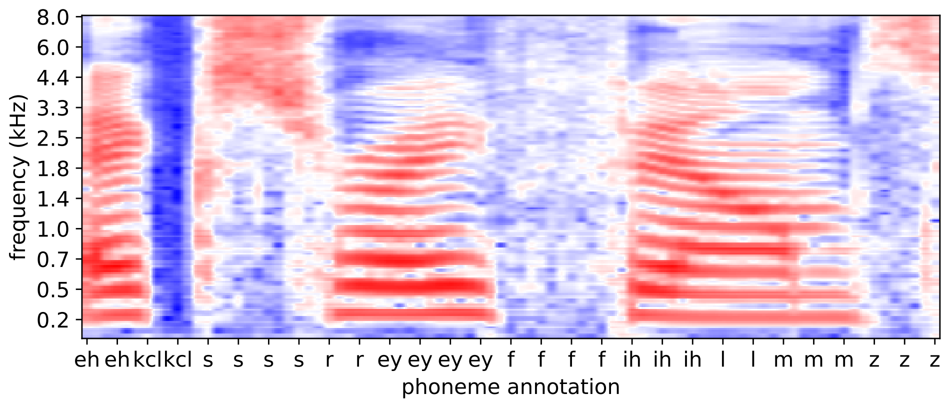


Figure 2.10: Mel spectrogram from the TIMIT data set including the phoneme annotations for every time step. The audio shows the spoken words 'X-ray films'. Note that, in the visualization, the time dimension is stretched by a factor of 4 to allow for readable axis labels. That means, one 'pixel' in the plot is 4 times as wide as high.

Alternatively, ASR can also be performed for a phonetic transcription. This means, instead of predicting letters, the sequence of sounds is predicted. To describe sounds, there exist phonetic alphabets which are sets of symbols that describe common sounds, typically in an individual language. The symbols used to describe these sounds are called phonemes. One popular alphabet is the International Phonetic Alphabet (IPA) [94] but there are different alphabets available, for example, CMUDict [23] or ARPABET [167]. An extension of the ARPABET is used in the TIMIT dataset [79], which we use in this thesis. In contrast to letters, phonemes can be associated with certain time frames in an audio recording because they describe sounds. Figure 2.10 shows an exemplary mel spectrogram from the TIMIT data set and the phoneme annotation for each time step. The entire sequence is additionally annotated with the letter transcription 'X-ray films'.

2.4.2 Automatic Speech Recognition

In this work, we use convolutional ASR, and therefore focus on background about this approach. Usually, for two-dimensional data like images, CNNs with two-dimensional filters are used. As speech is a one-dimensional signal, 1D CNNs are a suitable choice in speech processing. A 1D CNN can operate directly on a waveform or on a spectrogram [76, 77, 82]. Further, for operating on one-dimensional data, CNNs with one-dimensional filters are preferred over 2D CNNs because of the lower computational complexity [77]. Using CNNs for speech recognition is not uncommon [70, 20] but is often combined with models from traditional ML or other DL models. For example, such hybrid models involve Hidden Markov Models [2, 164] or RNNs [184]. Besides ASR, CNNs are also used for other speech-related tasks, for example learning spectrum feature representations [25] or speech emotion recognition [10].

Nevertheless, ASR can also be performed by other model types like RNNs [108] or LSTM [139]. More recently, there are successful ASR systems based on the popular Transformer architecture [72, 37]

Often, ASR models are not used in isolation to produce a good transcription but the output is refined by knowledge about natural language [28]. For example, DNNs that are trained to learn structure of language, so-called language models, can be applied to the output of an ASR model to correct spelling mistakes or grammatical errors [93].

2.5 Electroencephalography (EEG)

In this work, we take inspiration from how neuroscience analyzes and visualizes brain activity. Specifically, we adapt two approaches from EEG data analysis. EEG is a technique to measure the electric activity of the brain [106]. To this end, several electrodes are placed on the scalp, typically as a cap such that the electrode locations are at specified positions of the head (illustrated in Figure 2.11 left). Neurons in the brain communicate via small electrical signals, which are detected and amplified by the EEG electrodes on the scalp. The measured EEG signals are therefore a sequence of electrical signals at each of the electrodes.

In simplified terms, the brain activity can then be visualized by projecting the measured activity at each electrode to a schematic top-view of the head, indicating the activation strength by different color values, like shown in Figure 2.11 on the right. This visualization is called a topographic activation map [113]. In reality, this mapping is not directly possible because the

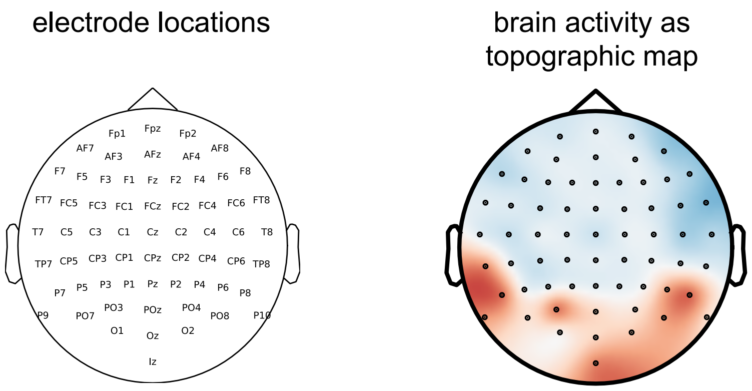

electrode locations
brain activity as topographic map
Fp1 Fpz Fp2
AF7 AF3 AFz AF4 AF8
F7 F5 F3 F1 Fz F2 F4 F6 F8
FT7 FC5 FC3 FC1 FCz FC2 FC4 FC6 FT8
T7 C5 C3 C1 Cz C2 C4 C6 T8
TP7 CP5 CP3 CP1 CPz CP2 CP4 CP6 TP8
P7 P5 P3 P1 Pz P2 P4 P6 P8
P9 PO7 PO3 POz PO4 PO8 P10
O1 Oz O2
Iz

Figure 2.11: Brain activity visualization with EEG. The left image indicates the positions of the electrodes. On the right, the measured brain activity is mapped to the electrode locations and interpolated in the regions without an electrode.

measured values contain a lot of noise. Because EEG is a non-invasive techniques where the electrodes are placed on the outside of the head, there is much signal interference from the skull and the neurons themselves. In addition, the brain processes many signals at the same time which makes it difficult to isolate, what task or sensory stimulus the recorded activity is associated with.

To obtain informative EEG measurements, the Event-Related Potentials (ERPs) technique is commonly used [112]. ERPs aim to isolate activity that is related to a certain event, also called a stimulus. To this end, EEG recordings are repeated multiple times for the same stimulus, called trials. The multiple recordings across the trials are then averaged. Due to the alignment at the stimulus, variations that are introduced by measurement noise or by activity that is not related to the stimulus are removed by the averaging. An illustration of the ERPs technique is shown in Figure 2.12 By applying this approach to all electrode locations, we obtain denoised event-related activity which can then be projected on the top-view representation of the head as in Figure 2.11.

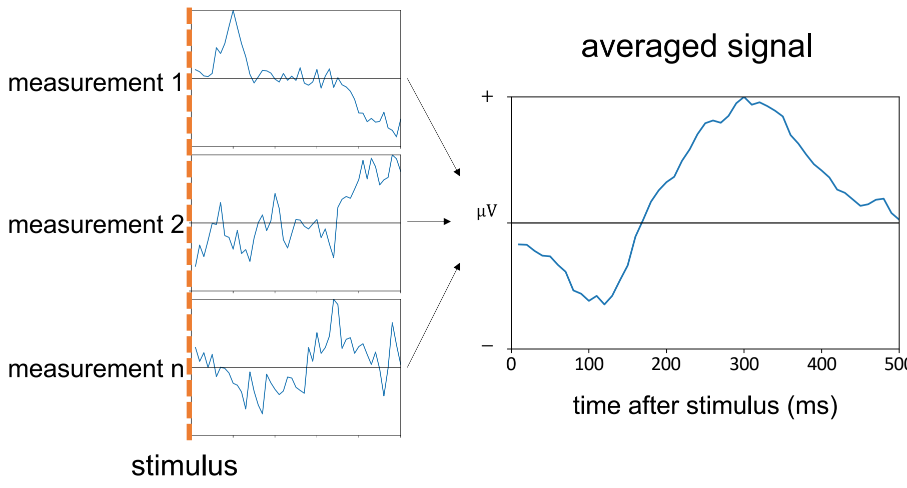


Figure 2.12: The Event-Related Potentials technique. EEG recordings across multiple trials for the same stimulus are averaged to obtain a stimulus-specific activation. Each individual measurement contains a lot of noise due to measuring the neural activity from outside the skull and due to the brain performing many processes simultaneously. Averaging multiple measurements for the same stimulus removes the noise while the consistent activity related to the stimulus is retained. The figure illustrates multiple measurements of a single electrode and the averaging result. In practice, this approach is applied to measurements of all electrodes, respectively.

3 Related Work

This thesis addresses the topic of DNN model evaluation. While basic evaluation techniques as described in Section 2.2.2 focus on whether the model performs well, our aim is to further understand better how it solves its task.

Model introspection is the process of analyzing or visualizing internal structures or processes of computational models. This is of particular interest in DL models as these work as black-boxes [194]. Research on elucidating DL model internals has gained traction recently and several introspection techniques have been proposed. However, most research is done in the field of computer vision due to the ease of visual inspection of image data [196, 162, 9].

In this chapter, we introduce relevant research in the field of DL model introspection and how this thesis relates to existing work in this field.

3.1 Terminology

Commonly, techniques that provide insight into structures or processes in DNNs is also referred to as explainable Artificial Intelligence (XAI) [59]. The term indicates that methods in this field of research lead to models that are better explainable. However, this is not the case for many techniques, including this thesis, that aim to provide means to interpret models that are not explainable themselves. We prefer to use the term 'introspection' because it reflects better that we provide insights into the model without making it inherently more explainable. In psychology, introspection refers to accessing one's own thoughts and feelings. From this perspective, the term is still not optimal for a analysis of DNNs performed by a person.

Based on the difference between implementing inherently more explainable models and explaining black-box models after they are trained, two research directions can be distinguished. Ante-hoc methods address creating models that perform their task in a human-interpretable way. Post-hoc introspection applies methods to inherently not interpretable models to gain insight into how they solve their task. In this thesis, we focus on post-hoc techniques.

In the following sections, we present different approaches on performing post-hoc model introspection. Commonly, these techniques are grouped by their scope [120]. According to their scope, introspection techniques can be categorized as local or global. Local methods explain single decisions and global approaches aim to explain a model comprehensively. However, the boundaries are not clear. On the one hand, analyzing results of local methods for many examples can give more global information about the model. On the other hand, global approaches that are applied to only parts of a model, for example a layer, are not data-dependent but also not entirely comprehensive. As the categorization by local and global scope is not always clear, we choose to group the techniques by commonalities of their approaches.

3.2 Inspecting Model Weights

A simple way of inspecting internal structures of a DNN is to visualize its learned weights. In fully-connected neural networks, each neuron in the first hidden layer is connected to all values in the input. To inspect the pattern that a particular neuron in this layer responds to, the weights of these connections can be visualized. To this end, the weight values are plotted according to the position of the corresponding values in the input [140] (Figure 3.1a). For image data or spectrogram inputs, the weights can be visualized as image, too (Figures 3.1a and 3.1b). In time-series data like speech recordings, weight visualization follows the one-dimensional structure of the input. In deeper layers, neurons receive input signals from neurons in the preceding hidden layer, however, neurons within the same layer are not interconnected and therefore have no informative ordering. Hence, in deeper layers, plotting the connection weights cannot be interpreted by visual inspection.

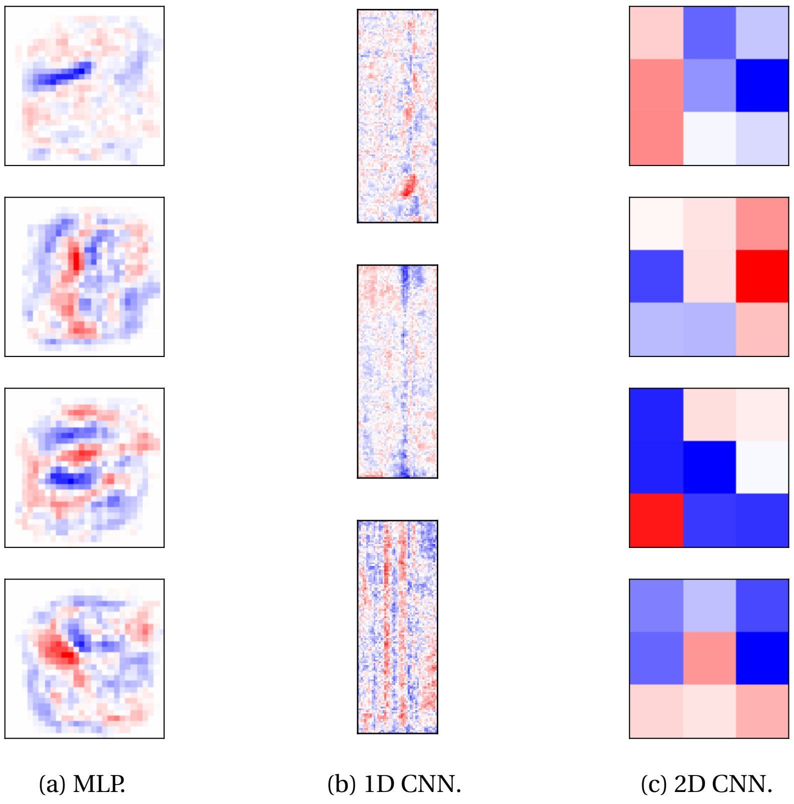


Figure 3.1: Weight visualization for different models. All subfigures use a 0-symmetrical color scale from blue over white to red. The range of the color scale is chosen for each subfigure individually. (a) Weights in the first layer of a MLP trained on MNIST. Each subfigure represents the connections of all input neurons to one particular output neuron in the first layer, reshaped to the size of the input 28 × 28 px. It is possible to interpret some features, like particular strokes at digit-specific positions. For example, the bottom weights might detect the upper left curved stroke of an eight. (b) 1D-convolutional filters in the first layer of a spectrogram-based ASR model. Each figure represents all filters stacked on top of each other, in the same order as the input channels. This ordering allows to identify patterns in the weights. For example, the top set of filters might detect changes of pitches in certain frequencies (diagonal shapes) and the other two detect changes across all frequencies, typical for certain sounds like a "t" or transitions from silence to a spoken word. Note that the filter size is relatively large with 48 time steps. Smaller kernel sizes would be more difficult to visually interpret. (c) Different 2D-convolutional filters in a CNN trained on Cifar10. The small size strongly limits the possibility of interpreting their function.

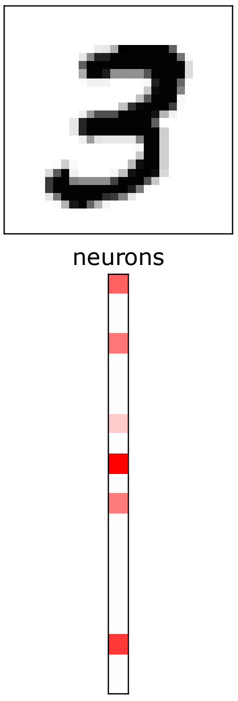
In CNNs, weights are applied as convolution operations with usually very small filters. For example, 3×3 is a common size of filters in 2D CNNs. For such small filters, it is almost impossible to understand their learned features only by visual inspection of the plotted weights (Figure 3.1c). In some cases, if the filters are large enough or if information can be connected between the channels, there is the possibility of interpreting the information, like for the shown 1D CNN example in Figure 3.1b.

3.3 Inspecting Model Activations

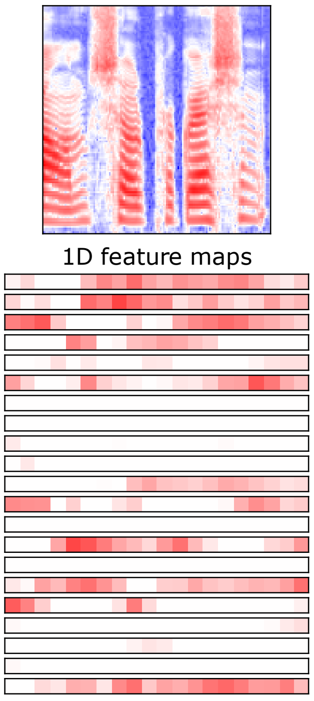
As visualizing weights in most CNNs, and generally in deeper layers, is not interpretable, activations can be visualized instead. In CNNs, the activations are arranged as feature map which can be interpretable to some extent, as shown in Figure 3.2c. Each position in the feature map only shows how strongly this convolutional filter activates for the corresponding region of the input. Therefore, feature maps only reveal to which part of the input a filter is responding but do not visualize the pattern which this filter detects [194]. However, with increasing depth, feature maps become more sparse as the detected patterns become more complex such that they also are not directly interpretable anymore, as illustrated in Figure 3.2d. In other model types like MLPs or 1D CNNs, activations are generally not interpretable by plotting them as shown in Figures 3.2a and 3.2b.

3.4 Feature Visualization

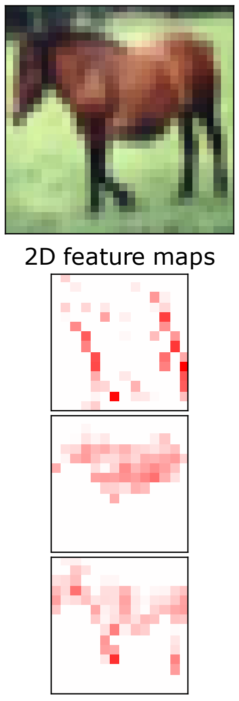
A common way for visualizing the features learned in deeper layers is to create an input which maximally activates individual neurons or sets of neurons [194, 33, 124]. Such input is obtained by an optimization procedure similar to the DNN training. When training a model, parameters of the network are updated to decrease the prediction error. For feature visualization, input values are updated to increase the activation values of a target set of neurons. In CNNs, complete feature maps are typically used as the target set of neurons. For brevity, we refer to the obtained inputs as “optimized inputs”. Optimized inputs can differ substantially from instances in the training data, making them difficult to interpret. For a human, optimized inputs of images often look unrealistic, as in the basic



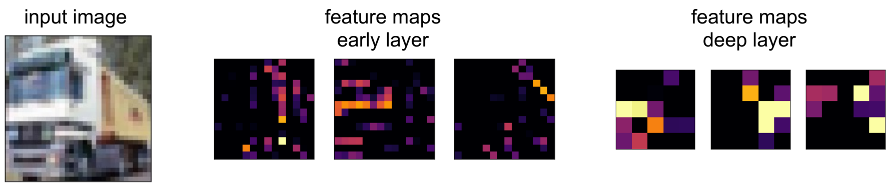
(a) MLP.



(b) 1D CNN.



(c) 2D CNN.

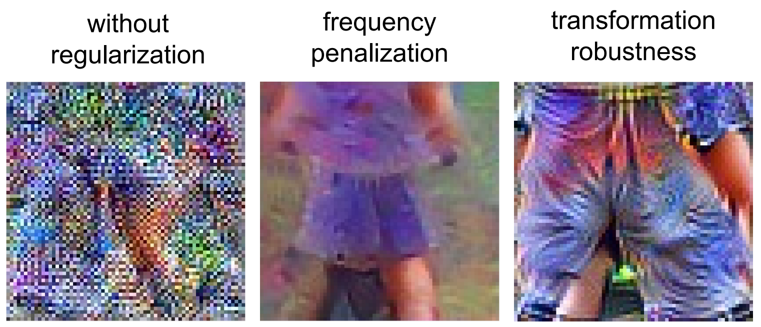


(d) 2D CNN feature maps in different layers. Early layer feature maps still resemble shapes that allow to interpret some detected features. Deeper layers activate for more complex features such that only the location, where it is detected, is visible.

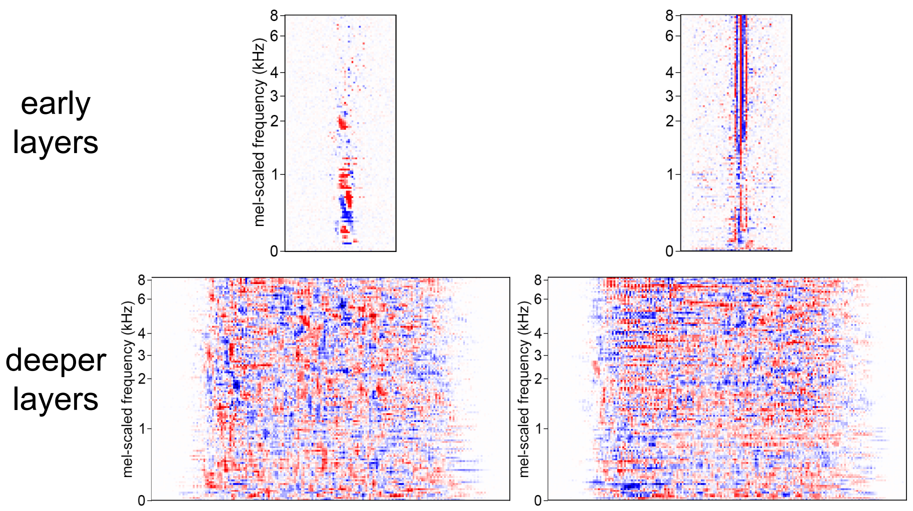
Figure 3.2: Activation visualization for different models (a – c) and in different layers of a 2D CNN (d). Feature maps of (d) use a different color map to emphasize the focus on layers instead of models. (a) Neuron activations in a MLP trained on MNIST. There is no spatial relation between the neurons and the image, therefore, it is impossible to obtain useful insight by visual inspection. (b) 1D feature maps in a 1D CNN that processes spectrograms of audio data. They allow for interpretation based on temporal relation, for example, that some feature maps activate for silence and others for time frames with sound. (c) 2D feature maps in a 2D CNN trained on Cifar10. The spatial relation to the input allows to connect the active positions to the detected pattern. For example, the first feature map seems to detect vertical edges and the second one might activate for brown color.

feature visualization shown on the left in Figure 3.3a. In the speech domain, optimized inputs often do not sound like natural speech or do not look like spectrograms of natural speech [85, 86], as shown in Figure 3.3b.

Regularization This problem can be tackled by applying regularization techniques to the optimization. Regularization adds constraints which, in addition to increasing activation of neurons of interest, encourage the opti-



(a) Feature visualization for an image recognition model with and without regularization. Regularization techniques help to obtain more natural patterns in comparison to an unregularized maximization of activations. Created with Tensorflow lucid [137, 158].



(b) Feature visualization for an ASR model in different layers with frequency penalization. While in early layers, patterns resemble patterns of spectrograms, the visualized features in deeper layers are highly unnatural.

Figure 3.3: Feature visualization results for exemplary data modalities.

mized input to be more similar to natural data. For example, natural data typically has few high-frequency information. To avoid such patterns, the optimization can be regularized with a L_1 regularization (compare Equation (2.3)) or adding blur to the inputs during optimization. Figure 3.3a shows an example of an optimized input with applying such so-called frequency penalization. As another example, the activation of the neuron that is supposed to be maximized should not depend on the exact location of the features in the optimized input. Hence, applying input transformations during the optimization helps in generating more natural features. An optimized input that is regularized by performing image transformations is shown in Figure 3.3a on the right. Such weak regularization techniques for frequency penalization [111, 132] and encouraging transformation robustness [124] are easy to implement but still tend to not look entirely natural.

Stronger regularization techniques encourage optimized inputs to be even more similar to data examples. The most basic approach is to identify examples in the data that highly activate a certain neuron [177]. However, this is limited to only showing existing examples. More flexibly, the distribution of the data can be learned with an unsupervised model like a Generative Adversarial Network (GAN) [133], an AE [134] or, more recently, diffusion models [47]. This learned distribution can be used as regularizer in the feature visualization by penalizing if the optimized input deviates from the learned distribution. Although stronger regularization leads to more natural visualizations, they might not represent the learned features well. For example, additionally minimizing the distance to a data example encourages the optimized input to be more natural but makes it impossible to detect whether a filter also reacts to unrealistic patterns.

Remarks Feature visualization can be considered a global introspection approach as it describes the model behavior independent of a given input example. However, as discussed before, the result only describes a part of the model and is not describing the model behavior in its entirety.

There is a relation between feature visualization and a concept called adversarial examples [44]. Both approaches optimize an input to change a neuron activation. Adversarial examples aim to minimally alter an input such that the output of the model changes. The target is to decrease the neuron activation of the correct output neuron. Feature visualization has the opposite optimization target because its aim is to increase a particular neuron's activation.

3.5 Saliency Maps

The strategy of another type of typical introspection techniques is to explain how a prediction for a single input example is made. To this end, such methods quantify how relevant each individual value in the input example is for the prediction [33, 171, 196, 173, 162, 73, 160]. In this sense, they are DL-specific successors of methods that measure feature importance in ML, like Local Interpretable Model-agnostic Explanations (LIME) [153] and SHapley Additive exPlanation (SHAP) [107]. LIME approximates a model's predictions with a linear surrogate model. To this end, the technique perturbs the input data, observes how these changes alter the output and uses this information to estimate the input feature importances. SHAP assigns importance by investigating different combinations of input features to derive their individual contributions to the importance, adapted from the concept of Shapley values [51] in game theory. While LIME and SHAP can be applied to DNNs, their results are not easy to interpret as the techniques are tailored towards interpretable input features like tabular data or texts. DL, however, typically is applied to data with large numbers of individually not interpretable features.

Focusing again on the DL-specific methods, they generally follow the strategy of first computing feature importance values, also called relevances. To visualize the obtained relevance values, they are commonly plotted as heat map overlayed on top of the input, particularly for image(-like) data. This heat map of relevance values is referred to as a saliency map [171]. An example is shown in Figure 3.4.

There is no definite method to compute relevance because the black-box nature of DNN models makes it impossible to exactly determine the correspondence between individual input values and the output of the model. Therefore, various techniques for obtaining relevance values for saliency maps have been proposed. A simple saliency map can be obtained by computing the derivative of the output value with respect to each input value. The resulting relevance describes how sensitive the output value is to

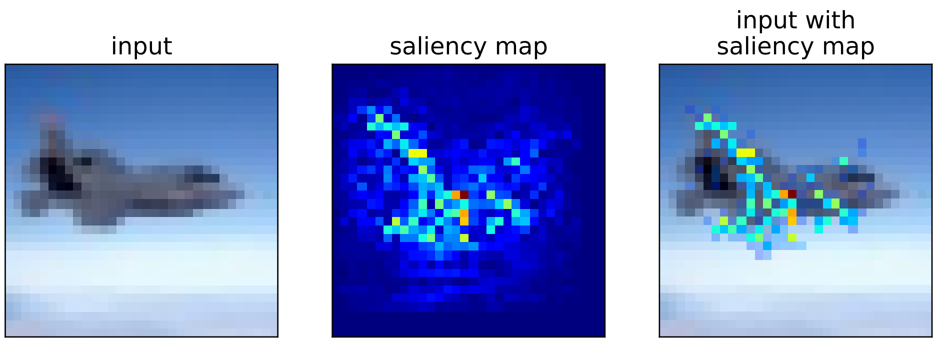


Figure 3.4: Saliency map for showing prediction-relevant parts of the input. The saliency is computed by sensitivity analysis, that is, the gradient of the predicted output ("plane") with respect to each pixel value, using the absolute gradient value. The saliency map is colored by a rainbow color map from blue to red (0 to maximum absolute value). Finally, the saliency map is superimposed on the input, with transparency of absolute gradients of less than 0.3. The sensitivity-based saliency map is noisy but it highlights meaningful parts of the object while showing less prediction-relevance of the background.

changes of each individual value of the input. Accordingly, this approach is called sensitivity analysis [33]. Other saliency map techniques, for example, approximate to invert the network (deconvnet [196]), use a decomposition approach (layer-wise relevance propagation (LRP) [9]) or combine sensitivity analysis with feature map activation (Gradient-weighted Class Activation Mapping (Grad-CAM) [162]). In the following paragraphs, we present common approaches to compute saliency maps. We provide exemplary saliency maps after the description of the techniques and refer to the respective figures in the individual paragraphs.

Gradient-Based Saliency Maps Gradient-based saliency maps focus on the gradient of the output class with respect to the input, with sensitivity analysis [33] as the simplest approach. Variations, for example, mask negative gradients during backpropagation (guided backpropagation [171]) to obtain relevance values. "Gradient × Input" [168], as the name of the method suggests, computes the element-wise product of relevances from sensitivity analysis and the input values to emphasize relevance of high input values. Integrated Gradients [175] does not only compute gradients for the original input, but for multiple interpolations between a baseline (zeros or random noise). Then, it approximates the integral over the gradients of these interpolated inputs as relevance values.

This category of saliency maps assigns relevance to every input value but is often spurious and can therefore be difficult to interpret. A saliency map created with Integrated Gradients is shown in Figure 3.5b as representative example of this category. We see that while the relevances are related to the features of the shown building, the saliency map is very noisy.

Perturbation-Based Saliency Maps While gradient-based techniques focus on sensitivity of the output to small changes in input values, perturbation-based saliency techniques apply larger changes to the input. Similarly, they then determine the effect on the model's output to obtain relevances. For occlusion sensitivity [196], patches of the input are replaced by zeros or random noise. Relevance is therefore obtained for an entire patch. To obtain a higher resolution of relevance values, overlapping patches can be investigated and the corresponding relevances merged together as a saliency map. Of course, more overlap of the patches requires more time to compute and can limit the method's applicability when wanting to create saliency maps for many examples. A generalized approach of occlusion sensitivity is feature ablation, which is not restricted to patches but uses combinations of individual features. However, for data with many features, it is infeasible to sample a representative amount of combinations. Instead of replacing the features with zeros or noise for occlusion, a set of input examples can be provided among which the features are permuted [39]. This way, the features are changed to more realistic values but the relevances are highly dependent on the choice of inputs.

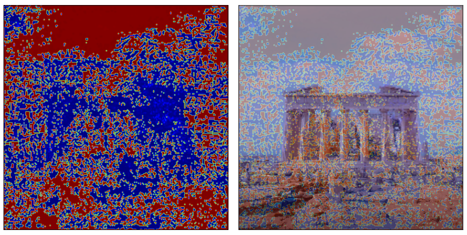
We present an exemplary perturbation-based saliency map using occlusion sensitivity with patch size of 25×25 px for an image of size 224×224 px in Figure 3.5c. The salient regions appear reasonable but the patched approach leads to inaccuracies, as visible by salient regions extending higher than the top of the building because the patch included both the important building feature and the sky on top of it.

Decomposition-Based Saliency Maps Another set of methods combine gradient and activation information and aim to propagate values from the output neuron backwards through the network. LRP [9] follows the idea of starting with a fixed total relevance (typically) at the final output neuron of interest, then distributing this value backwards through the layers depending on the activation and weights of the previous layer. There are different rules in LRP that allow to customize how total relevance is distributed from one layer to the previous one. For example, this controls whether to separately consider negative and positive influences and how to weight both against each other. Deep Taylor Decomposition [122] follows a similar concept as LRP but performs the decomposition (that is, the backwards distribution of relevances) using a Taylor series expansion. DeepLift [169] propagates relevance back in a network by contrasting to a reference output. This means, the technique requires to define a reference input which is then used to explain differences in the output in relation to it. The reference input is a neutral state, therefore, the focus shifts to differences to an expected state. For example, in a set of input images with a common background, an image of the isolated background is suitable as a reference input. Due to the contrasting approach, DeepLift is then expected to describe the relevance without the less interesting contributions from the common background.

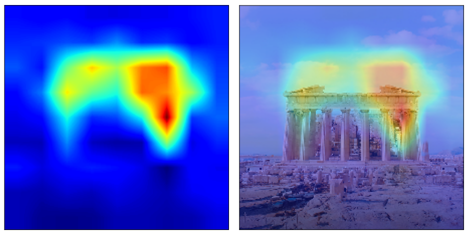
In Figure 3.5d, we show an exemplary LRP saliency map. It has a high resolution of relevant features that appear reasonable as they are related to the building structure. However, it is not very specific about which exact features might be relevant for the model. Note that, in contrast to the other shown techniques, LRP obtains positive and negative relevance values.

Class Activation Map-Based Saliency Maps Thirdly, there are saliency map techniques which primarily use activations of CNN feature maps as indicators of relevance. Most popular are CAM-related methods that are based on projecting feature map activations of a particular layer for a single input example to the input space. The earliest variant of CAM [197] uses feature maps immediately before the output layer. It computes a weighted sum of these feature maps, with the connection weights from the feature maps to the predicted output as weighting factors. Later, Grad-CAM was introduced which, instead of the weights, used the average gradient of the feature maps as weighting factor [162]. Further, they suggested to multiply Grad-CAMs with the result from guided backpropagation to make use of its higher-resolution information. Averaging the gradient over feature

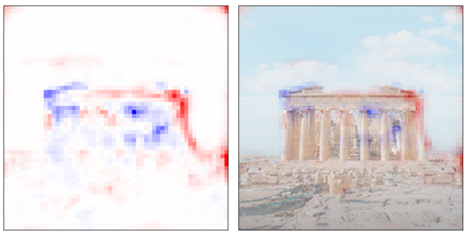
(a) Input example



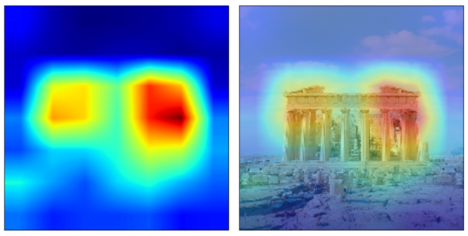
(b) Integrated Gradients



(c) Occlusion Sensitivity



(d) LRP



(e) Grad-CAM

Figure 3.5: Saliency maps for a image classification created with different methods. VGG16 was used as a prediction model using 'triumphal arch' as the target class. The LRP saliency map was created with code from the original authors [121], the others using the tf-explain library [117]. The LRP saliency map is shown with a 0-symmetric blue-white-red color scale as relevance values can be negative and positive. All others use a color map blue-green-yellow-red from 0 to the maximum value. Gradient-based techniques like (b) Integrated Gradients typically produce noisy saliency maps that might show details but are not easy to interpret. Occlusion Sensitivity (c) and Class Activation Mapping (CAM)-based saliency maps (e) are more easy to interpret as they highlight relevant regions instead of individual values. Occlusion Sensitivity is less accurate as it observes patches while CAM-based methods upscale feature maps. Decomposition approaches like LRP (d) again focus on details but are less noisy than gradient-based methods. However, in some cases, they can overemphasize individual features.

maps in Grad-CAMs favors large over smaller features. To represent smaller features better, the successor called Grad-CAM++ adjusts the weighting depending on the size of the detected feature [19]. The aforementioned CAM approaches assign weights for entire feature maps. To account for the possibility that parts of different feature maps are relevant for the prediction, LayerCAM performs Grad-CAM-like relevance computation for every feature map position [65]. Moreover, LayerCAM proposes to combine the saliency maps from different layers to represent coarse localization information and detailed relevance simultaneously. Finally, Score-CAM is a CAM approach that does not require gradients [189]. To compute weights for feature maps, it first computes the feature maps for the example of interest. Then, each feature map is used to mask parts of the input, occluding parts for which the activation is low, and the model output is obtained for this occluded input. The difference between the output scores of the original input and the occluded input is used as the weight for the respective feature map when computing the weighted average.

CAM-related methods are often considered visually pleasing due to their smooth regions. These, however, are due to upscaling the activation from the feature map size to the input size using interpolation techniques. Therefore, the shown relevances in the saliency map can be not directly related to the individual input values they are superimposed on. An exemplary Grad-CAM saliency map is shown in Figure 3.5e. While the low level of detail is beneficial for coarsely localizing prediction-relevant areas in the input, it is difficult to draw conclusions about whether meaningful features are used for the prediction.

Applicability of Saliency Maps to Audio Data Saliency maps are easy to interpret if the input data are interpretable by visual inspection themselves. Hence, they are suitable for image data but less applicable to sensor data like speech or EEG recordings. Using spectrograms, it is possible to use saliency map methods, too [11, 182, 149], as shown in Figure 3.6. However, interpreting the results is only possible for experts with domain knowledge about reading spectrograms.

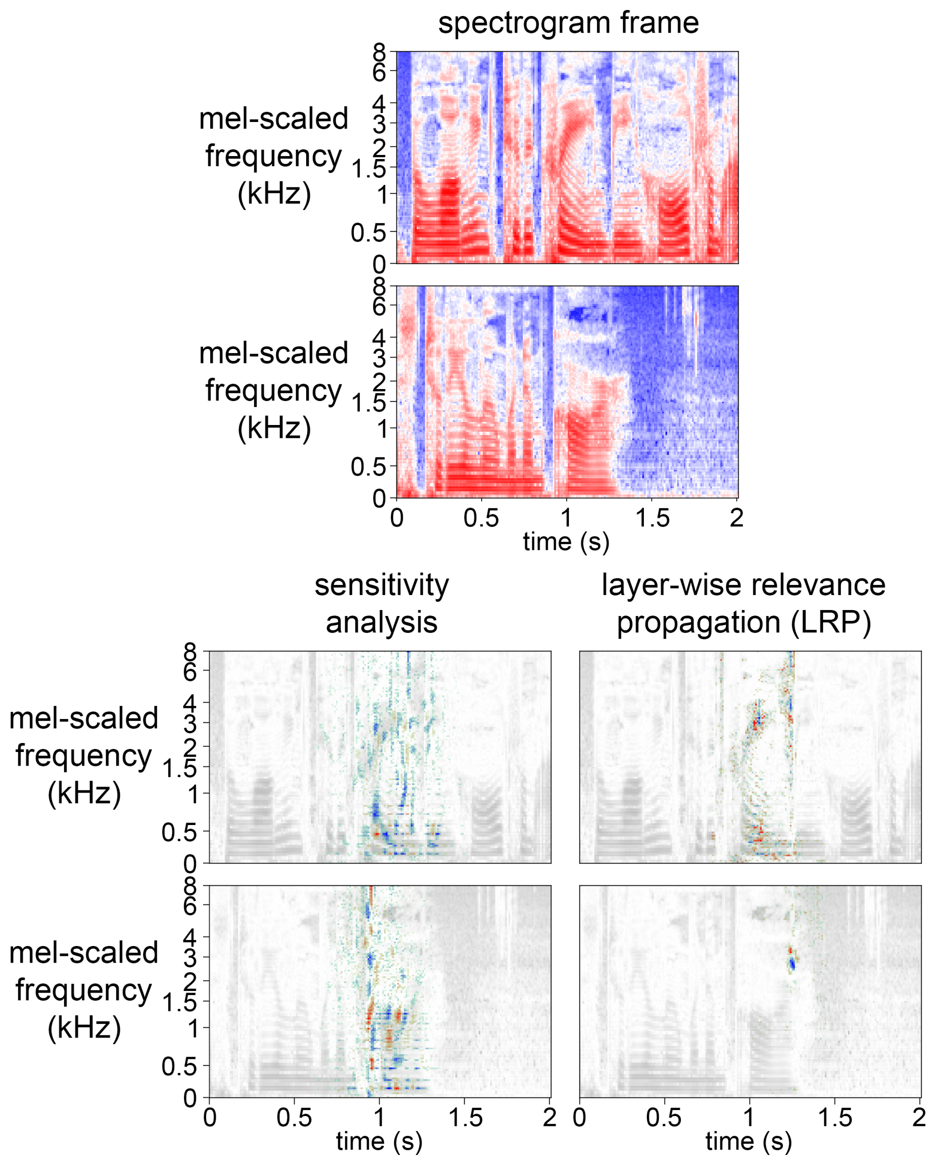


Figure 3.6: Saliency maps for spectrograms using sensitivity analysis and LRP. Both methods highlight features of the input that relate to some speech features. As in the general discussion of saliency maps, sensitivity analysis is rather noisy and identifies many features as relevant while LRP is more specific but potentially overemphasizes features.

Remarks As a local introspection technique, saliency map methods work on single examples. This makes it hard to assess the model comprehensively. Furthermore, methods for computing saliency maps need to be chosen carefully as some can be misleading. Adebayo et al. [4] demonstrate this issue by investigating how saliency maps change when setting layer weights to random values. They found that, for some methods, the explanations barely change even when completely randomizing network weights. Similarly, Nie et al. [135] explained why backpropagation-based visualizations can be weakly related to network predictions. Sixt et al. [172] identified similar behavior of more recent methods for computing saliency maps.

In this work, we use sensitity analysis-based saliency maps as a tool to identify prediction-relevant parts of inputs. However, we only use it as an auxiliary technique within our analysis pipeline.

3.6 Analyzing Data Set Representations

More comprehensive insight into DNNs can be provided by analyzing representations of different classes using the complete data set. For example, Alain and Bengio [6] train linear classifiers on intermediate representations to quantify their representative power for the prediction. Such linear classifiers are the basis for the research of Kim et al. [71] who derived vectors that represent user-defined concepts. Fiacco et al. [38] introduced functional neuron pathways, which are co-activated sets of neurons identified through PCA. Representational similarity can also be investigated through Canonical Correlation Analysis (CCA) [123] or by clustering of class-specific neuron activations [128]. In speech, the latter type of analysis was conducted for MLPs for speech-to-phoneme prediction [128, 129, 127] and convolutional ASR [85, 88].

Clustering-based activation similarity analyses can be visualized as cluster maps, which are similarity heat maps that are sorted by a hierarchical clustering, as shown in Figure 3.7 for an exemplary MNIST CNN.

Further, there also are several interactive tools to investigate representations and learned features. The What-If Tool [190] focuses on data with interpretable features and provides several interactive visualizations to help understanding the model performance and how predictions change when altering features in a specific input. One main functionality, as the name of the tool indicates, is the possibility of investigating “what if”-hypotheses. An interactive visualization allows to see feature differences between out-

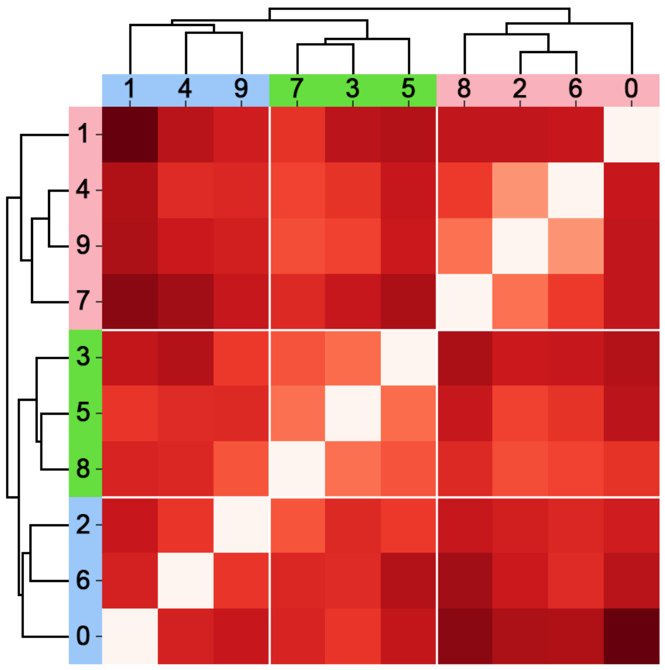
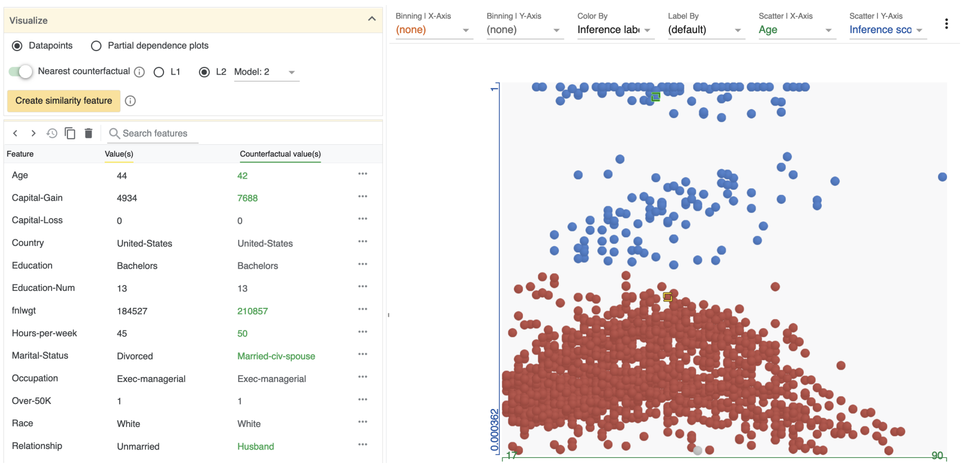


Figure 3.7: Visualizing relations of groups by creating a cluster map based on their activations for the classes of an MNIST CNN. Here, average activations over all examples from the same class are computed and their similarity is computed with a hierarchical clustering using Euclidean distance. White indicates identity and darker shades of red indicate less similarity.

puts of multiple examples and, for a selected example, shows minimal changes necessary to obtain a desired different prediction. The user can also manually change feature values and visualize how this affects the model prediction. Figure 3.8 shows the view of this interactive visualization. However, the What-If Tool does not provide information about model internals.

An activation atlas [17] is a visualization technique that employs feature visualization in a more data-oriented way. There are two general methodological differences to feature visualization as described before. First, to obtain targets (neurons, feature maps or layers) to optimize for, activation atlases investigate which parts of a network are active for a specific input example. Second, instead of maximizing the activation of the target, they optimize an input such that the model shows activations that are similar to those of the original input. This allows activation atlases to produce



Feature	Value(s)	Counterfactual value(s)
Age	44	42
Capital-Gain	4934	7688
Capital-Loss	0	0
Country	United-States	United-States
Education	Bachelors	Bachelors
Education-Num	13	13
fnlwgt	184527	210857
Hours-per-week	45	50
Marital-Status	Divorced	Married-civ-spouse
Occupation	Exec-managerial	Exec-managerial
Over-50K	1	1
Race	White	White
Relationship	Unmarried	Husband

Figure 3.8: Datapoint editor in the What-If Tool. It shows prediction scores of earning more than 50 thousand dollars per year for multiple examples. We selected a negatively classified entry close to the decision boundary and let the tool show the nearest example that is positively predicted. The tool allows to manually change features and shows the changes in the model prediction. Created with the What-If Tool's example notebooks [141].

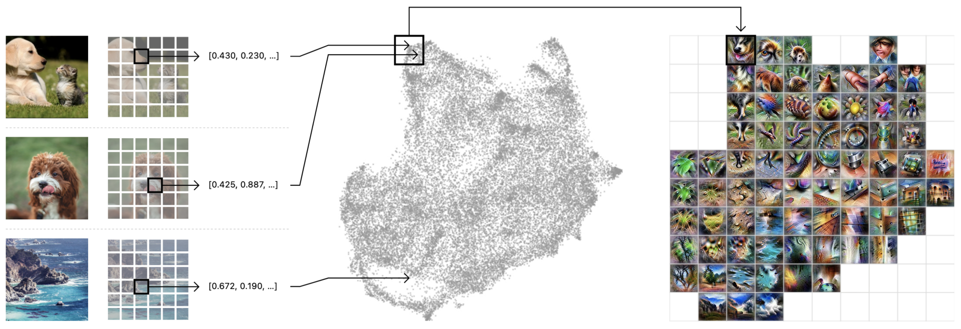


Figure 3.9: Activation atlas for InceptionV1 representations based on a set of natural images. The final activation atlas shows a grid of feature visualization results. Each grid position represents a set of image patches that are nearby in a two-dimensional projection (UMAP) of DNN activations of the patches. While the visualized features are not entirely natural, it is possible to interpret some categories. For example, bottom grid positions might represent buildings with sky, plant features are visible in the left grid positions and some animal features like snouts or eyes in the top regions. Figure taken from the original publication [17].

a visualization of which patterns learned by the models are used to perform certain predictions. Further, as relevant features do not always span the entire input, they apply their approach on patches of inputs, in their case, image patches. The activation atlas is created in a multi-local way. Activations are obtained for patches of many images. A UMAP projection (compare Section 2.3) is used to obtain a two-dimensional representations of the patches according to their activation similarity. Next, they divide this representation into an equally-spaced grid and compute the average activations in each cell of the grid. This average activation is then used as target activation values for the feature visualization approach. The final activation atlas is then a grid-view of visualized features which represent the activation space for a large number of data examples. An illustration of the activation atlas computation process is shown in Figure 3.9. As visible from the figure, activation atlases inherit disadvantages from the feature visualization component. For image data, it is possible to interpret some of the visualized features, but they are generally unnatural images which complicates drawing conclusions about the model's inner workings from the results.

Summit [56] and Neurocartography [144] are related visual analysis tools for inspecting features learned by a DNN and how they relate to each other. Summit focuses more on the classes while Neurocartography is more neuron-focused.

Summit [56] provides a high-level overview of how classes are related according to their activations. To this end, they represent the images by obtaining activations for each example, applying global max-pooling to the feature maps and keeping the 3 % highest active feature maps to represent the activity for this example. This aggregated activation information is then used to project the classes to a two-dimensional space using UMAP. Further, it provides a detailed view of the identified features by allowing to interactively look at feature visualizations of the features related to a class and related features in adjacent layers. In addition to the feature visualization, Summit also shows input examples that maximally activate the respective feature. Figure 3.10 shows an exemplary view from the Summit tool for inspecting the ladybug class.

Neurocartography [144] aims to visualize features maps and their relations. To this end, they first determine conceptual similarity of the feature maps.

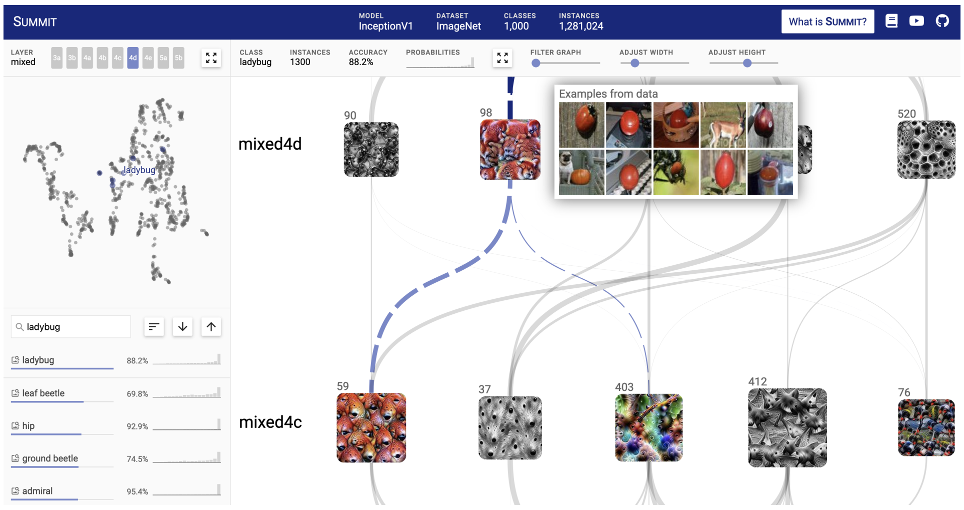


Figure 3.10: The Summit tool for visualizing features that are associated with the selected class. We selected the ladybug class and looked into a feature that appears to be activated by round red objects. Figure created with the Summit tool [57].

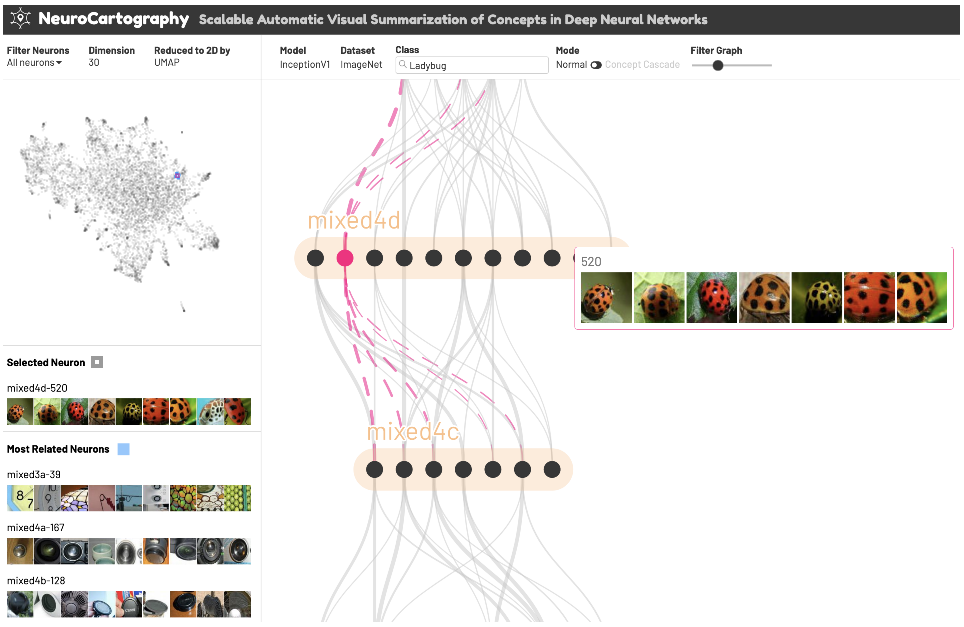


Figure 3.11: The Neurocartography tool for visualizing feature maps and their relations to each other. We selected to show features related to the ladybug class. While some features reasonably belong to the ladybug class, others are not that clearly related. Figure created with the Neurocartography tool [143].

They follow the idea that feature maps are related if they activate for similar parts of the same example. Therefore, they cluster the feature maps by their similarity across many input examples. Similar to Summit, there is a UMAP projection as a high-level overview, in this case of the feature maps according to their activation similarity. The detailed view of Neurocartography shows a view of the neuron clusters and how they are related between the layers. For each of the clusters, it shows input examples that highly activate it, as well as for the most strongly related similar clusters. An example of using the Neurocartography tool to inspect features related to the ladybug class is shown in Figure 3.11

Representation analyses can also incorporate information from the previously described introspection techniques. For example, Spectral Relevance Analysis [95] uses LRP-based saliency maps and applies spectral clustering [188] to determine commonalities in the explanations instead of the activations.

Like saliency maps, methods that analyze representations of many examples from the data set rely on the provided data examples for their explanations. Behavior for inputs which are different from the training data distribution can be analyzed by using appropriate test data. It is even possible to cover the behavior for inputs for which the network prediction is ambiguous. To this end, a data set of adversarial examples can be generated, such that minimal changes in these inputs change the prediction of the network [44], using the optimization approach of feature visualization. However, it requires infeasibly huge amounts of additional data to cover all possible inputs. Therefore, even by analyzing data set representations, it is impossible to completely describe the behavior of the model.

Our approach also analyzes and visualizes representations. However, in contrast to most existing work, we focus more on the ease of visual inspection and less on highly detailed information about activation similarity of neurons. The closest to our work is the overview visualization of feature maps in Neurocartography. They share the idea of projecting neurons to a two-dimensional space based on their activation similarity across many examples. We, however, use a different approach of obtaining activity of the neurons, aim for a visually more appealing layout of the two-dimensional space and additionally show activity of the neurons in the 2D space by coloring them accordingly.

4 Data and Models

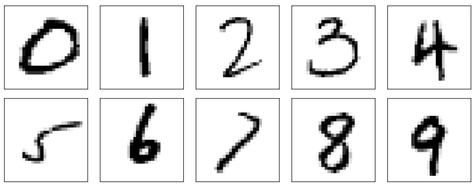
In this chapter, we describe data sets as well as model architectures and their training parameters that we use in this thesis.

4.1 Data

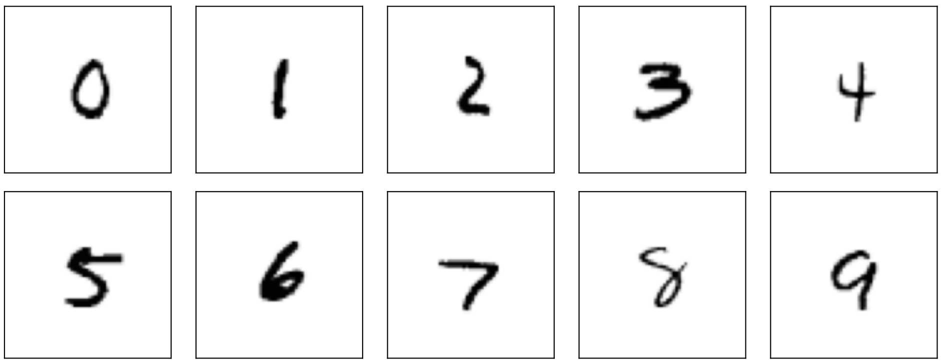
4.1.1 MNIST

MNIST [97] is a common benchmark data set for Machine Learning. MNIST contains grayscale images of handwritten digits from 0 to 9, which are of size 28×28 px, centered and normalized in scale, as shown in Figure 4.1a. There are $60,000$ and $10,000$ training and test data examples, respectively.

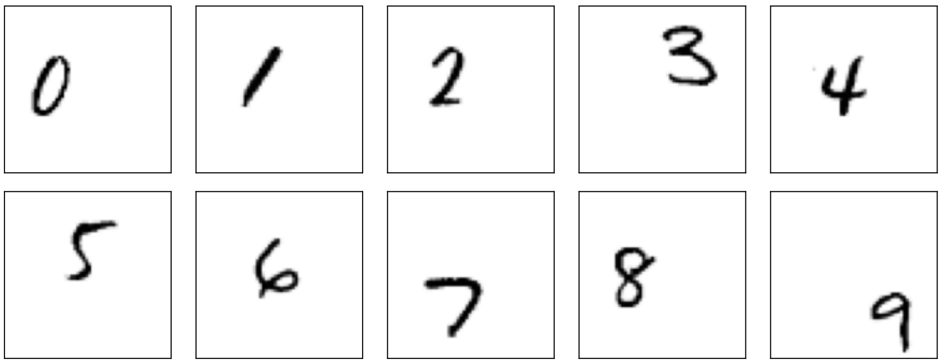
MNIST Variations For some evaluation experiments in this thesis, we create a version of the MNIST data set in which the written digits are not aligned at the center of the image. To this end, we randomly position each 28×28 px MNIST image in an empty 56×56 px image. For each image, we draw two random numbers between 0 and 28 and use them as the top left coordinate at which we insert the original image in the empty larger image, resulting in examples as shown in Figure 4.1c. With this approach, we remove the data property that all digits in the images are center-aligned. This allows us to investigate the applicability of our approach to objects of interest being at different locations in the input. To fairly compare this altered MNIST data set to the original one, we further create a data set that comprises the original MNIST images padded to a size of 56×56 px (see Figure 4.1b). We refer to these two MNIST variations as shifted MNIST and padded MNIST, respectively. As the MNIST variations are based on the original MNIST data set, they contain the same number of training and test examples.



(a) MNIST with resolution of 28 × 28 px



(b) Padded MNIST with resolution of 56 × 56 px



(c) Shifted MNIST with resolution of 56 × 56 px

Figure 4.1: Exemplary inputs for each class of MNIST and of our padded and shifted variations. Shifted MNIST positions the original 28 × 28 px images randomly in an empty 56 × 56 px image while padded MNIST retains the alignment at the center. The variations of MNIST are useful to investigate the influence of alignment on the results of our methods.

4.1.2 Fashion MNIST

A similar data set to MNIST is Fashion MNIST [191]. Both data sets contain the same number of training and test images, share a common image size and are grayscale. Fashion MNIST, in contrast, contains of images of 10 different clothing items or accessories (see Figure 4.2). The objects in Fashion MNIST are more complex than the digits in the MNIST data set. Moreover, some sets of classes are similar which makes them more difficult to distinguish than digits, for example, there are three shoe categories that are more similar to each other than to the rest of the classes.

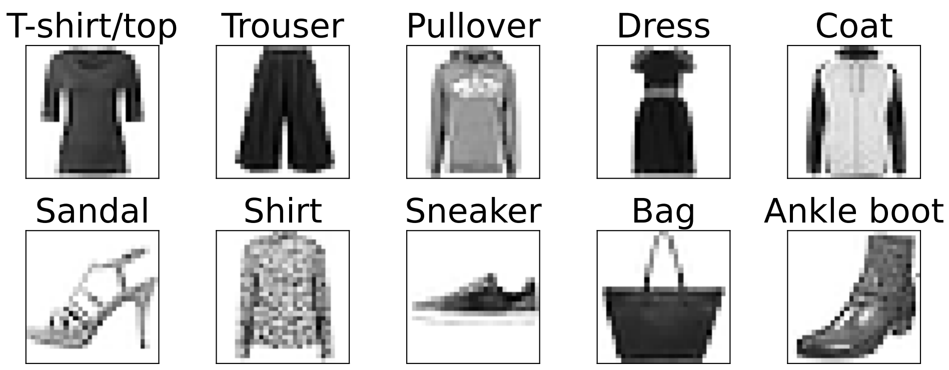


Figure 4.2: Exemplary inputs for each class of Fashion MNIST. While sharing the dimensionality and data set size of MNIST, the classes in Fashion MNIST have more complex features and more variation within the classes. Moreover, some classes are clearly more similar than others. For example, shoe classes are similar as well as 'Shirt' and 'Pullover', while 'Dress' or 'Bag' are intuitively better distinguishable.

4.1.3 Cifar10

Unlike MNIST and Fashion MNIST, the images in Cifar10 [83] are not in grayscale. There are 50,000 training images and 10,000 test images. Each image's dimension is $32 \times 32 \times 3$. One image contains only one object and there are 10 different classes in this data set. Objects are not always in the center of an image and an object might not be fully present in an image. Figure 4.3 shows an exemplary input from each class.

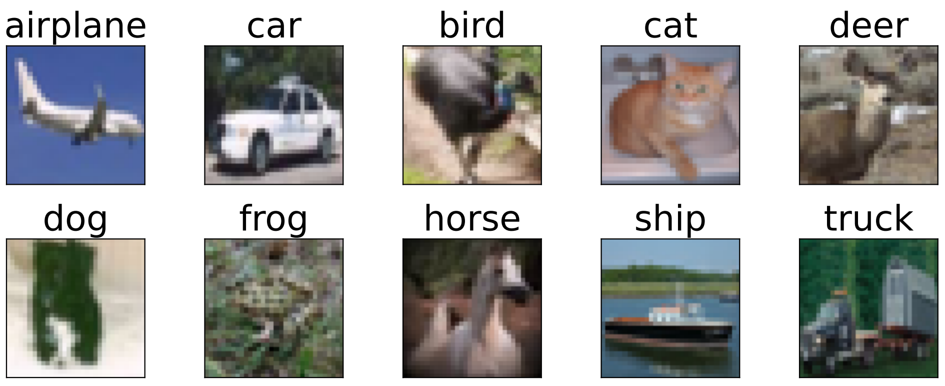


Figure 4.3: Exemplary inputs for each class of Cifar10. This data set is more complex than MNIST and Fashion MNIST as it comprises colored images of slightly larger size and of natural objects in different environments. However, the small resolution makes it difficult for a human to detect the correct category in several cases.

4.1.4 LibriSpeech

The LibriSpeech Corpus [142] is a common benchmark data set for ASR. It consists of about 1000 hours of recordings from the audio book domain, sampled at 16 kHz. The authors split their data set into sets based on how much word error rate they observe on a pre-trained model. Recordings with a low word error rate were used for a so-called “clean” data set. This “clean” training data contains 363.6 h of audio from 921 speakers and the test data 5.4 h from 40 speakers. Each recording is provided with a transcript of the spoken sentence.

We included LibriSpeech in this overview despite not using it in this thesis specifically. However, we show in our previous work (Krug et al. [88]) that our approach is applicable to large-scale audio data sets like LibriSpeech. In this thesis, we focus on TIMIT because it provides phoneme annotations and therefore allows for a more in-depth evaluation of our technique.

4.1.5 TIMIT

Typical audio benchmark data sets like the LibriSpeech corpus do not contain phoneme annotation, which limits the analysis to investigating the letter prediction. This is not optimal because letters in texts are often ambiguous as to how they are pronounced. In our previous work [88], we addressed this issue by obtaining a phoneme mapping through a letter-to-phoneme translation model. This mapping transcribed words to phonemes, which allowed analyzing how the letter prediction model responds to phonemes. Still, it was not possible to compare the predicted letters with a ground truth occurrence of phonemes in the speech recording.

For an in-depth analysis of our method, we require a data set which already provides annotations of the phonemes in the input data. Therefore, to be able to perform this kind of analysis, we train our ASR models on the TIMIT data set [41] because it provides phoneme annotations.

TIMIT is a small speech corpus in English language, containing 6,300 speech recordings. Each of the 630 speakers recorded 10 out of 2,345 unique sentences, but the distribution of how often each sentence was recorded is not uniform. In particular, TIMIT includes two sentences which are recorded for each of the 630 speakers, causing these sentences to be massively overrepresented. To avoid overfitting on these instances, we exclude these sequences. We train model using the provided data split, but exclude the majority of the recordings of said two sentences. For the NAP analyses, we use available examples from both the training and test data.

The letter transcriptions are readily available for training letter prediction models. Because the training of our model does not require targets for each time step, we can use texts as targets without mapping the letters to time steps. To obtain phoneme prediction targets, we need to adapt the provided phoneme annotations in TIMIT. These annotations provide a mapping of phonemes to time steps as a sequence of phonemes and their duration until the spoken phoneme changes. To convert this annotation into a target which can be used with the loss function of our model, we discard the information about the exact time duration and only use the phoneme sequences as transcription targets. Due to the small number of examples and speakers in TIMIT, the models trained on this data set do not generalize well in terms of ASR performance. However, this is no limitation for this work because we do not aim for high speech recognition capability but for demonstrating and evaluating our model introspection technique.

All data are preprocessed to log mel power spectrograms using the librosa library [114], applying an STFT with window size of 512 (32 ms) and hop size of 128 (8 ms) at 16 kHz and projecting the frequency bins to 128 mel-frequency bins. For NAP analyses, we split the spectrograms into frames of 206 steps (2 s) corresponding to the receptive field size of the model, that is, the time frame which the model uses for a single prediction. This results in about 330,000 spectrogram frames.

4.1.6 FairFace

FairFace [68] is a balanced data set of images of people from different age groups, races and binary genders. It was created with the aim of detecting and mitigating bias in algorithmic decision making models. It comprises 108,501 facial images that were primarily obtained from the YFCC-100M Flickr data set [181].

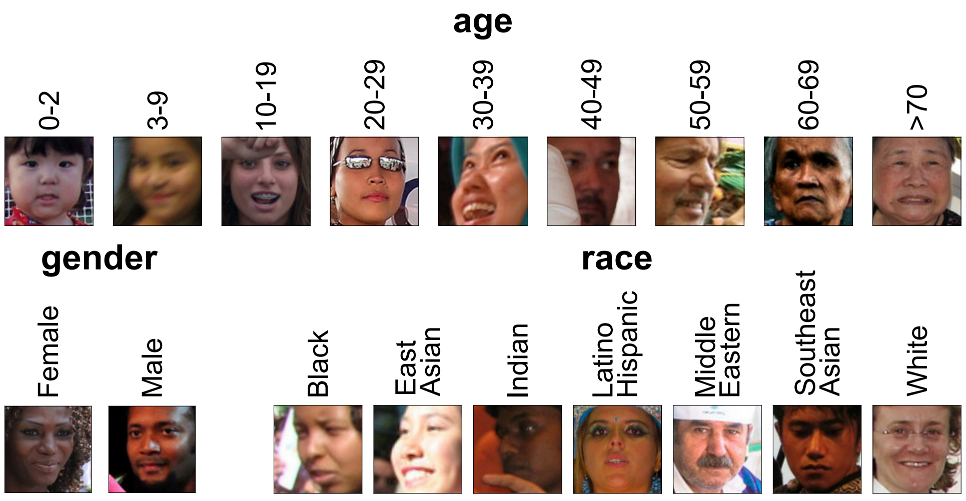


Figure 4.4: Exemplary inputs for each class of FairFace using variables age, gender and race, respectively. This data set is particularly useful for investigating biases in models as it is a balanced data set in terms of common sensitive variables. As each example is annotated with gender, age and race, there are intersections between the groups in different sensitive variables.

Each images is annotated with four properties: the age category (9 classes), the binary gender and two race classes either in four high-level categories (for example "Asian") or in seven more low-level categories (for example, dividing "Asian" into "East Asian" and "Southeast Asian"). Examples for the categorization by age, gender and race are shown in Figure 4.4. We use this data set only as an evaluation data set and use the validation data provided by the authors.

4.1.7 Data Overview

In Table 4.1, we provide an overview of the used data sets, including their modality and dimensionality as well as the number of included training and test examples.

name	modality	dimensions	# training examples	# test examples
MNIST	grayscale image	28×28	60,000	10,000
padded MNIST	grayscale image	56×56	60,000	10,000
shifted MNIST	grayscale image	56×56	60,000	10,000
Fashion MNIST	grayscale image	28×28	60,000	10,000
Cifar10	RGB image	32×32	50,000	10,000
TIMIT	audio as log mel spectrogram	$t \times 128$	3,696	1,358
FairFace	RGB image	224×224	86,745	10,955

Table 4.1: Overview of data sets. Note that the dimensions of TIMIT are dynamic as each recording has a different duration and hence a different number of time steps t in the log mel spectrogram.

4.2 Models

4.2.1 MNIST and Fashion MNIST Classifiers

We train a simple MLP on the MNIST data set. The MLP has one fully-connected hidden layer of 128 neurons and uses ReLU [130] activation. The input images are flattened before providing them to the model. For both MNIST and Fashion MNIST, we train a simple CNN, respectively. The CNN has two 2D-convolutional layers with kernel size 3×3, stride 2 and 128 filters, both using ReLU activation. The fully-connected classification layer takes the flattened feature maps of the second convolutional layer as input. During training, we use dropout and spatial dropout for fully-connected and convolutional layers, respectively, with a dropout rate of 0.5. An overview of the architectures is given in Table 4.2. Both models are trained with Tensorflow [1], batch size 32 for 20 epochs using the Adam optimizer [74] with default parameters and categorical cross-entropy as the loss function.

model	layer type	kernel size	stride	padding	neurons/ filters	params
MLP	Flatten					0
	Fully-Connected				**128**	**100,480**
	Fully-Connected				10	1,290
						Σ =101,770
CNN	Conv2D	3×3	2	valid	128	1280
	Conv2D	3×3	**2**	valid	**128**	**147,584**
	Flatten					0
	Fully-Connected				10	46,090
						Σ =194,954

Table 4.2: Architecture details of the MLP and CNN models trained on MNIST and Fashion MNIST. The respective layers for creating the topographic map visualizations in the experiments in Section 7.3.3 are highlighted in bold print. We do not train an MLP on Fashion MNIST as it does not obtain good performance.

Model performances measured by total and per-class accuracy are reported in Table 4.3. For the MNIST data set, both MLPs and CNN perform well on the training set and also generalize well to the test data. The CNN shows about 2 % higher accuracy than the MLP. The CNN-based Fashion MNIST classifier has approximately 12 % less accuracy than the one trained on MNIST, which indicates slight underfitting. Nevertheless, the performance on Fashion MNIST training and test data is similar, indicating that the model generalizes well to unseen data. From the per-class accuracies, we observe especially low performance for the 'Shirt' and 'Sneaker' classes, likely because of their similarity to other classes. 'Shirt' is similar to 'Pullover' as both are long-sleeved clothing items for the upper body and 'Sneaker' is one of three shoe categories in the Fashion MNIST data set. The performance could have been increased by using a more complex model but we aim to use the same architecture for both data sets for comparability reasons.

	MNIST MLP		MNIST CNN			Fashion MNIST CNN	
class	training	test	training	test	class	training	test
	96.95 %	96.07 %	98.68 %	98.25 %		84.79 %	83.59 %
0	97.07 %	98.27 %	99.19 %	99.29 %	T-shirt/top	82.70 %	79.90 %
1	97.56 %	97.27 %	98.97 %	98.59 %	Trouser	97.57 %	97.80 %
2	98.79 %	97.38 %	99.09 %	97.87 %	Pullover	72.42 %	71.20 %
3	96.12 %	96.44 %	98.74 %	98.51 %	Dress	82.37 %	80.90 %
4	96.03 %	95.32 %	99.08 %	98.88 %	Coat	86.29 %	83.50 %
5	94.32 %	94.17 %	97.49 %	98.54 %	Sandal	99.20 %	99.30 %
6	99.17 %	96.87 %	99.48 %	98.33 %	Shirt	68.35 %	62.30 %
7	95.87 %	92.90 %	99.54 %	98.35 %	Sneaker	64.92 %	68.10 %
8	98.51 %	97.74 %	97.52 %	96.92 %	Bag	98.55 %	98.10 %
9	95.94 %	94.15 %	97.50 %	97.22 %	Ankle boot	94.71 %	94.80 %

Table 4.3: MNIST and Fashion MNIST model performance, reporting accuracy across the entire data set and per class.

4.2.2 Padded and Shifted MNIST Classifiers

Our CNNs for classifying the larger-size MNIST variations are similar to those used for the original MNIST, but use an additional convolutional layer to reduce the feature map size, as shown in Table 4.4. Note that we use a high number of feature maps of 128 in each layer. Fewer feature maps would suffice to learn the task but a higher number of feature maps is beneficial for our topographic map visualization. As for the MNIST CNN, we use ReLU activation as well as spatial dropout for the convolutional layers with a dropout rate of 0.5, and use the same optimizer and loss function. Also like for the MNIST CNN, we train the padded and shifted MNIST classifiers with batch size 32 for 20 epochs using the Adam optimizer with default parameters and categorical cross-entropy as the loss function.

Model performances measured by total and per-class accuracy are reported in Table 4.5. Both models have a high accuracy across all classes and generalize well to the test data. The shifted MNIST classifier has around 2 % less accuracy than the padded MNIST classifier. Although the features for detection do not differ between the data sets, they can vary in location in

model	layer	layer type	kernel size	stride	padding	neurons/ filters	params
CNN	1	Conv2D	3 × 3	2	valid	128	1280
	2	Conv2D	3 × 3	2	valid	128	147,584
	3	Conv2D	3 × 3	2	valid	128	147,584
	4	Flatten					0
	5	Fully-Connected				10	46,090
							Σ =342,538

Table 4.4: Architecture details of the CNN models that we trained on the shifted and padded versions of MNIST.

the shifted MNIST data set. Therefore, the classification is a slightly more difficult task for the model to solve and leads to the lower accuracy.

	padded MNIST		shifted MNIST	
	CNN		CNN	
class	training	test	training	test
	98.46 %	98.18 %	96.91 %	96.00 %
0	99.50 %	99.69 %	98.49 %	97.35 %
1	99.62 %	99.56 %	98.59 %	98.77 %
2	98.38 %	97.48 %	96.06 %	94.86 %
3	97.48 %	97.92 %	97.86 %	96.93 %
4	98.78 %	97.96 %	97.56 %	96.74 %
5	96.72 %	97.76 %	95.19 %	96.08 %
6	99.79 %	98.02 %	98.55 %	96.66 %
7	99.27 %	98.64 %	97.61 %	94.94 %
8	97.62 %	97.02 %	93.76 %	92.61 %
9	97.19 %	97.52 %	95.11 %	94.75 %

Table 4.5: shifted and padded MNIST model performance, reporting accuracy across the entire data set and per class.

4.2.3 VGG16

VGG16 [170] is a pre-trained CNN model that is commonly used as feature extractor for downstream applications like image recognition DNNs. An overview of the architectures is given in Table 4.6. All convolutional and fully-connected layers use ReLU activation except for the output layer.

model	layer type	kernel size	stride	padding	neurons/ filters	params
VGG16	Conv2D	3×3	1	same	64	1,792
	Conv2D	3×3	1	same	64	36,928
3	**MaxPooling2D**	2×2	**2**	**valid**		**0**
	Conv2D	3×3	1	same	128	73,856
	Conv2D	3×3	1	same	128	147,584
6	**MaxPooling2D**	2×2	**2**	**valid**		**0**
	Conv2D	3×3	1	same	256	295,168
	Conv2D	3×3	1	same	256	590,080
	Conv2D	3×3	1	same	256	590,080
10	**MaxPooling2D**	2×2	**2**	**valid**		**0**
	Conv2D	3×3	1	same	512	1,180,160
	Conv2D	3×3	1	same	512	2,359,808
	Conv2D	3×3	1	same	512	2,359,808
14	**MaxPooling2D**	2×2	**2**	**valid**		**0**
	Conv2D	3×3	1	same	512	2,359,808
	Conv2D	3×3	1	same	512	2,359,808
	Conv2D	3×3	1	same	512	2,359,808
18	**MaxPooling2D**	2×2	**2**	**valid**		**0**
	Flatten					0
	Fully-Connected				4096	102,764,544
21	**Fully-Connected**				**4096**	**16,781,312**
	Fully-Connected				1000	4,097,000
						Σ =138,357,544

Table 4.6: Architecture details of the VGG16 model. The respective layers for creating the topographic map visualizations in the experiments in Section 7.4 and Appendix A.5 are highlighted in bold print.

4.2.4 Wav2Letter-Based Models

We use a fully-convolutional architecture for ASR. This architecture is based on Wav2Letter (W2L) by Collobert et al. [24], which is a 1D-convolutional architecture consisting of 11 layers, trained using the Connectionist Temporal Classification (CTC) loss [151]. This model can transcribe variable-length inputs of speech recordings to sequences of characters.

In this work, we train the model using the TIMIT data set [41] to transcribe speech as spectrogram to sequences of letters or phonemes. Furthermore, we made a few changes to the original architecture. As one adaptation, we changed the number of convolution filters in each layer to the closest power of two to improve computation efficiency on graphics cards [63]. Another difference to the original W2L model is that our architecture comprises one layer more in the stack of layers 2–9. We added one layer because a total of 12 layers allows for arranging information about all layers on a 2×6 or 3×4 grid. Although we do not make use of this property in this work, it makes the architecture a more convenient exemplary architecture for demonstrating visualization results. The parameters of the convolutional layers of the W2L architecture according to our adaptations are shown in Table 4.7. Before each convolutional layer, we use a batch normalization layer. In all layers except the output layer, we use ReLU as activation function.

Usually an ASR system would use a subsequent language model to improve the model's output. As we focus on analyzing how an ANN-based acoustic model processes speech, we do not apply additional post-processing to the outputs.

Model Variations

For our experiments, we use five different variations of the W2L architecture. The used parameters for the convolutional layers of all models are provided in Table 4.7.

The W2L architecture for letter prediction is our reference model. In our experiments, we continue to refer to the trained model as *W2L*.

For alignment evaluation (Section 5.3.3), we use W2L-based models that predict phonemes. One model is identical to the original architecture, but predicts phonemes. Accordingly, this model uses 62 output units in the output layer. We refer to this model as *Wav2Phoneme* (W2P). As we expect phoneme prediction to be an easier task than letter prediction, we also expect a model with fewer layers to be capable of performing this task. Therefore, we use a second phoneme prediction model that comprises only 5 convolutional layers and a final phoneme prediction layer, which we refer to as *W2P_shallow*. The used parameters of the convolutional layers are identical to the corresponding layers in the complete W2L architecture. We train both phoneme prediction models using the transcriptions generated from TIMIT phoneme annotations.

model name	layer	#convolution filters	kernel size	stride
W2L, W2L_TL_frozen & W2L_TL_finetuned	1	256	48	2
	2–9	256	7	1
	10	2048	32	1
	11	2048	1	1
	(output) 12	29	1	1
W2P	1	256	48	2
	2–9	256	7	1
	10	2048	32	1
	11	2048	1	1
	(output) 12	<u>**62**</u>	1	1
W2P_shallow	1	256	48	2
	<u>**2–5**</u>	256	7	1
	(output) <u>**6**</u>	<u>**62**</u>	1	1

Table 4.7: Overview of parameters of 1D-convolutional layers in the used models. The respective differences of W2P and W2P_shallow to the W2L architecture are highlighted in bold print with underline.

For evaluating experiments on representational similarity (Section 7.2.2), we train models such that there is an expected layer in which phonemes are best represented. To this end, we first train the W2P_shallow model to predict phonemes. From this model, we remove the final phoneme prediction layer (layer 6). On top of the pre-trained layers, we add layers to form the W2L model again, using letters as output. This approach is a transfer learning from a shallow phoneme prediction model (W2P_shallow) to a deeper letter prediction model. An illustration of this approach is shown in Figure 4.5. In the first step of transfer learning, we only train the added layers by disabling parameter updates of the pre-trained layers. As disabling parameter updates is commonly referred to as freezing, we call the resulting model *W2L_TL_frozen*. In this model, we expect the base layers to encode phonemes, while the top layers learn to map from the phoneme representation to letters. In the second transfer learning step, we unfreeze the base layers and perform a fine-tuning over all layers (*W2L_TL_finetuned*). Note that we do not fine-tune the batch normalization layers as this can lead to unlearning the pre-training information. We use the W2L_TL_finetuned model to investigate to which extent the fine-tuning process changes the pre-trained phoneme encoding in favor of better prediction of letters.

We train all W2L-based models with Tensorflow, batch size 16 for 300,000 batches using Adam optimizer with default parameters and the CTC loss. A summarized overview of all W2L model variations used in this work is shown in Table 4.8.

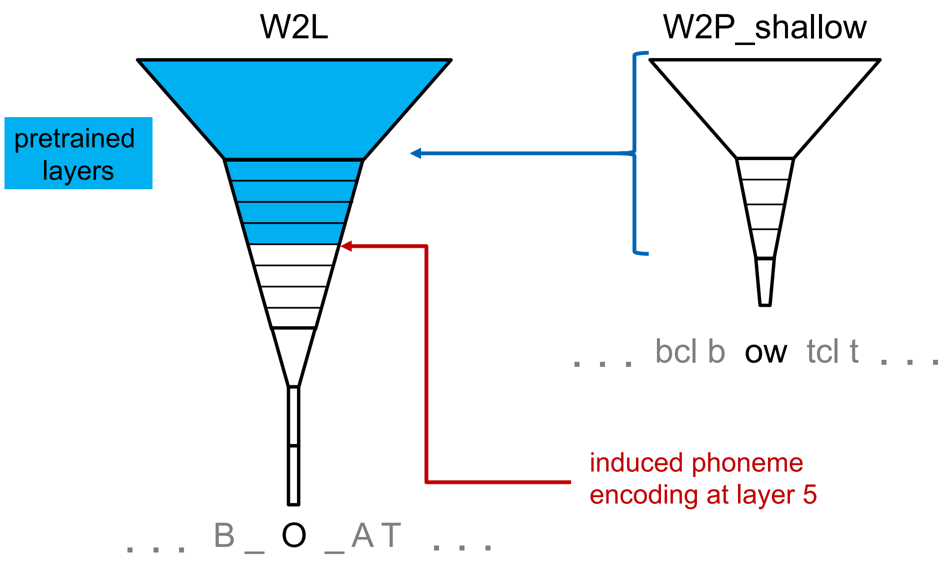

W2L
W2P_shallow
pretrained layers
. . . bcl b ow tcl t . . .
induced phoneme encoding at layer 5
. . . B _ O _ A T . . .

Figure 4.5: Illustration of the W2P to W2L transfer learning approach. A shallow model (right) is trained to predict phonemes and its initial pre-trained layers are used as base layers (blue) for the deeper W2L architecture (left).

model name	#layers	output type	initializing weights using model	used in experiments
W2P	12	phonemes	none	AE
W2P_shallow	6	phonemes	none	AE
W2L_TL_frozen	12	letters	W2P_shallow	RS
W2L_TL_finetuned	12	letters	W2L_TL_frozen	RS
W2L	12	letters	none	PS, RS, LP

Table 4.8: Overview of W2L-based models and which experiments they are used in. AE: Alignment Evaluation (Section 5.3.3), PS: Plotting SNAPs (Section 7.2.2), RS: Representational Similarity (Section 7.2.2), LP: comparing letter and phoneme representations (Section 7.2.2).

Performances of all W2L-based models are reported in Table 4.9. As the models perform a transcription, we compute the Levenshtein distance [98] between the target and the prediction sequence of phonemes or letters, respectively. We use the 'editdistance' package [180] for python to compare the distance values. Note that the number of edits can only be fairly compared if the models has the same type of outputs. For the phoneme prediction models W2P and W2P_shallow, we observe a higher training set performance of W2P compared to W2P_shallow. On the other hand, the test set performance of W2P is lower than that of W2P_shallow. This means, W2P_shallow is better generalizing to unseen data than W2P. This is likely related to the deeper architecture of W2P which is, as discussed above, too complex for phoneme prediction and therefore prone to overfitting. Our observations confirm this expectation. The letter prediction models share the same architecture but differ in the way they are trained. W2L has good performance on the training data but the largest test set errors. W2L_TL_frozen has worse performance on the training data but a better generalization capability than W2L. W2L_TL_finetuned shows the best performance among the letter prediction models, both for training and test data. The improved performance of W2L_TL_finetuned is likely also related to a higher number of total training steps because it is initialized with weights from W2L_TL_frozen, which in turn is partly initialized with weights from W2P_shallow.

Table 4.9: W2L-based model performances, reporting total edit distance and average edit distance per example. Training and test data comprise 3,696 and 1,358 examples, respectively.

model	training		test	
	# edits	edits per example	# edits	edits per example
W2P	555	0.15	21,457	15.80
W2P_shallow	2,184	0.59	20,545	15.13
W2L_TL_frozen	1,584	0.43	10,958	8.07
W2L_TL_finetuned	217	0.06	10,635	7.83
W2L	843	0.23	12,052	8.87

4.3 Data and Model Usage Overview

This section provides an overview of which models are used for which experiments and points to the respective sections. A tabular overview is shown in Table 4.10.

We perform our main evaluations with the W2L model and its variations (see Section 4.2.4) and the models trained on the padded and shifted MNIST data (see Section 4.2.2).

With the padded and shifted MNIST models, we discuss the effects of using our alignment technique in Section 5.3. Further, we use these models to investigate how well NAPs can be approximated with random subsets of the groups in Section 5.4. Finally, a large part of the quantitative evaluation of topographic activation maps in Section 6.3.3 is based on these models.

W2L and its variations are particularly useful for Section 5.3 as they allow us to perform an evaluation of our alignment procedure based on the TIMIT data set ground truth expectations for the alignment. For this evaluation, we use W2P and W2P_shallow. TIMIT, with predictions from W2P_shallow, also serves as an unbalanced data set for investigating NAP approximation of groups of different sizes in Section 5.4. Finally, using the letter prediction models W2L, W2L_TL_frozen and W2L_TL_finetuned, we show an exemplary NAP analysis of ASR models in Section 7.2.

The MNIST and Fashion MNIST CNNs (see Section 4.2.1) and the VGG16 model (see Section 4.2.3) are used for exemplary applications of topographic activation maps. With MNIST and Fashion MNIST and a simple CNN architecture, we demonstrate error detection in Section 7.3. Our bias detection experiments in Section 7.4 are based on the VGG16 model and the FairFace data set.

Lastly, in Chapter 8, we present multiple extended use cases using a very simple MNIST MLP model (see Section 4.2.1). We use it to demonstrate multi-layer visualization in Section 8.1, visual inspection of activations during the training process in Section 8.2 and the relation of model confidence and activity in Section 8.3.

data	model	used in
padded MNIST, shifted MNIST	3-layer CNN	alignment evaluation (Section 5.3) NAP approximation (Section 5.4) topographic map evaluation (Section 6.3.3)
TIMIT	W2P, W2P_shallow	alignment evaluation (Section 5.3) NAP approximation (Section 5.4)
TIMIT, (LibriSpeech)	W2L, W2L_TL_frozen, W2L_TL_finetuned	Analysis of an ASR model (Section 7.2)
MNIST, Fashion MNIST	2-layer CNN	topographic map use cases (Section 7.3)
(ImageNet), FairFace	VGG16	bias visualization with topographic activation maps (Section 7.4)
MNIST	1-layer MLP	multi-layer visualization (Section 8.1) training process visualization (Section 8.2) relation of confidence and activity (Section 8.3)

Table 4.10: Overview of models and where they are used in this thesis. Note that ImageNet was used to train VGG16 but we use a pre-trained model and therefore do not work with the data set directly.

5 Neuron Activation Profiles

In this chapter, we address our first research aim (RA1, Section 1.1). To characterize group-specific DNN responses, we introduce Neuron Activation Profiles (NAPs) which are an ERP-inspired averaging approach. The ERP technique characterizes brain activity for a specific event (stimulus) by averaging multiple EEG measurements (trials) that are aligned at the same stimulus. As the event is consistent across all trials, aligning the data at this stimulus and averaging the signals yields event-specific information [106]. Averaging of a random signal over multiple measurements, in contrast, returns the expected value of its random distribution. This way, in ERPs, variations in brain activity that are unrelated to the stimulus are removed from the measurements. In DNNs, activations of any hidden layer can be obtained directly, therefore, removing noise from the measurement is not necessary. Instead, in our ERP adaptation for DNNs, we average activations over multiple examples that belong to a common group. This way, the averaging removes variations that come from the individual examples while it retains the information that is common between the examples of the group. Like in ERPs, this averaging approach requires that the measurements are aligned properly. Therefore, in its essence, NAPs are computed by aligned averaging to obtain information that is common within a group of inputs.

Publications We introduced different aspects and improvements of NAPs in several publications for interpreting convolutional speech recognition models. First, we averaged activations of particular spoken words [84], assuming that the predictions provide suitable alignment positions. However, in the used model, the predictions were made with a non-systematic offset with respect to the input data. Therefore, we introduced an alignment technique that makes use of information about the prediction-relevance of input positions [85]. In this publication, we demonstrated the technique for the input data itself and later generalized the approach to any hidden layer activations [88]. Finally, we integrated the above approaches into a common NAP technique [87, 89]. Additionally, these final NAP publications improved the aligned averaging approach by masking out information that is irrelevant for the prediction. With this applied mask, we obtain NAPs that characterize both which activations are observed across the group and which of these activations are important for the prediction that the network makes. Our latest NAP publications referred to the technique as Gradient-Adjusted Neuron Activation Profiles (GradNAPs) and Saliency-Adjusted Neuron Activation Profiles (SNAPs), respectively, to emphasize the incorporated prediction relevance. In this thesis, we will use the term NAP to describe the general approach of which there are variations, for example, the masking of irrelevant information.

Chapter structure First, we describe the NAP approach and its variations in Section 5.1. After that, we evaluate the individual steps in more detail and discuss requirements and limitations of the applicability of the approach (Sections 5.2 – 5.4). Finally, we end with a summary of the chapter (Section 5.5).

5.1 Method

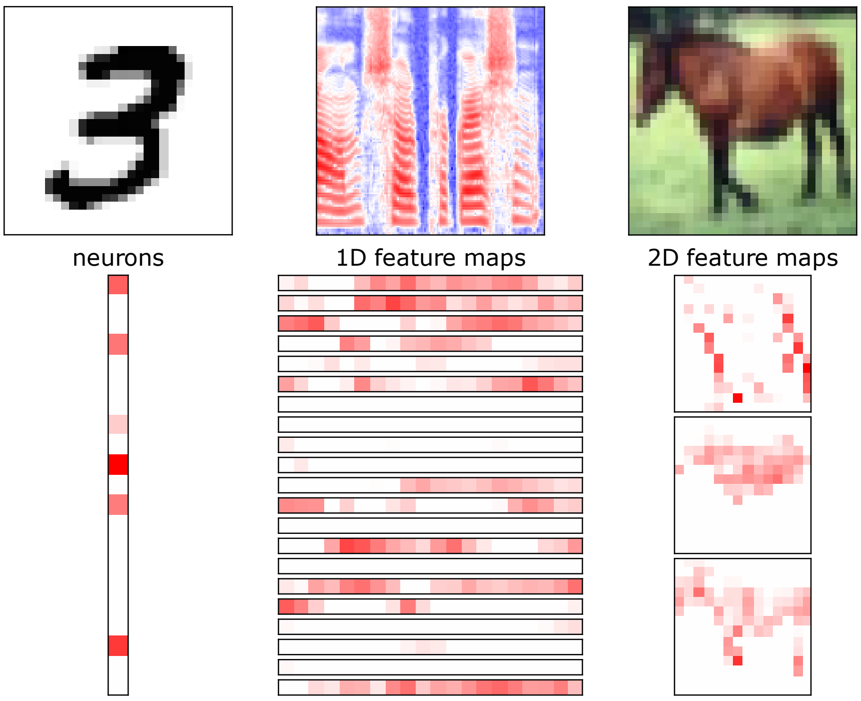
In this section, we describe the individual steps and variations of the NAP approach. We follow a conceptually logical order to introduce the steps, therefore, they do not reflect the order in which the steps are applied in. Section 5.1.9 gives a summarized overview of the entire approach and its variations in a chronological order.

To support the understanding of each step, we show exemplary results from specific data sets or layers as a visual aid. Nevertheless, each of the steps is applicable to hidden layer activations and to other data sets equivalently.

5.1.1 Obtain Activations

The core idea of NAPs is to obtain and analyze DNN activations for a large number of input examples. Obtaining the activation values is straightforward by performing a forward pass through the network and storing the outputs of each individual layer. We will refer to the outputs of each individual layer l, given the input example X_i as $A_l(X_i)$, using A to fit to the term activations. Further, we consider the input data itself a layer, such that we also consider the input values to be activations. That means, $A_l(X_i) = X_i$ for l being the input layer.

Depending on the model and layer, the activations A_l are of different dimensionality. For example, the input layer A_l follows the input dimensionality, activations of 2D-convolutional layers are three-dimensional and output layer activations are one-dimensional. Figure 5.1 shows exemplary activation patterns for an MLP (Figure 5.1a), 1D CNN (Figure 5.1b) and 2D CNN (Figure 5.1c).



(a) MLP. (b) 1D CNN. (c) 2D CNN.

Figure 5.1: Obtained activations for individual input examples from different data and model types. For demonstration purposes, only a small subset of neurons or feature maps is shown. In the bottom row, color intensity indicates higher neuron activation. (a) Neuron activations in an MLP trained on MNIST. (b) 1D feature maps in a 1D CNN that processes spectrograms of audio data. (c) 2D feature maps in a 2D CNN trained on Cifar10.

Because the range of values and dimensionality of each layer are almost always different, the largest instance for which a NAP can be computed is a layer. In principle, it would be possible to compute NAPs for smaller instances, like for feature maps. However, aiming for the most comprehensible characterization of the activations, we suggest to use layer NAPs.

Choosing Layers of Interest In practice, the term layer is not precisely defined. Some implementations treat the layer that applies the learned weights (for example, a convolutional layer) separately from the activation function, such that there are two individual layers. Other implementations treat the activation function as part of the layer. In more complex architectures, what appears to be an individual layer in the implementation can also be a custom building block comprising multiple layers. Moreover, some layers only scale the outputs of a preceding layer without altering its information content, like batch normalization layers. To reduce the usage of storage and computation resources, we suggest to only investigate a representative subset of layers. The applied activation function is the main reason for the expressive power of DNNs, therefore, we use the outputs after the activation function is applied. In the case of custom building blocks, if this block defines a conceptually related module, like a Dense Block [62] or an Inception Module [179], we recommend to only use the output of the block. If, in contrast, the building block comprises many layers, for example, when using a pretrained network as a fixed set of layers in a transfer learning task, it is beneficial to also investigate the contained individual layers. Finally, layers that do not change the information content during inference should be omitted, as the resulting NAPs would be redundant with those of the preceding layers. This includes, for example, batch normalization and dropout layers.

5.1.2 Define Groups

Because NAPs characterize network activations when processing groups of examples, the groups of interest first need to be defined. Consider a data set $\mathbf{X}$ that comprises n examples $X_i \in \mathbf{X}$, with $i \in 1,...,n$. A group $\mathbf{G}$ can be any subset of the entire data set $\mathbf{G} \subset \mathbf{X}$. For our approach, useful groups are supposed to have a conceptual similarity, for example, belonging to the same category or sharing a common feature. The most typical grouping of a data set describes a collection $\mathbb{G}$ of m groups, such that the groups are disjunct and that their union is the entire data set.

$$\text{grouping } \mathbb{G} = \{\mathbf{G}_j\}_{j=1\ldots m} \quad \text{with} \quad \mathbf{G}_j \cap \mathbf{G}_k = \emptyset, \forall j \neq k \quad \text{and} \quad \bigcup_{j=1\ldots m} \mathbf{G}_j = \mathbf{X} \tag{5.1}$$

In a multinomial classification task, an intuitive grouping is to use the annotated or predicted classes as groups. Each example X_i belongs to exactly one group, therefore, the set of groups corresponding to classes satisfies the definition of a grouping according to Equation (5.1). Groupings provide the most comprehensive way to define groups of interest because they encompass the entire data set and all examples are represented exactly once. This does not imply that every grouping is also useful, which will be discussed later in this chapter.

Note that our method is not limited to groupings according to Equation (5.1). Any set of groups can be investigated, including groups which partly intersect or that only cover parts of the entire data set. Section 5.1.7 discusses other useful groupings as well as applications where incomplete groupings or overlapping groups can be reasonable.

To explain the next steps, we continue to use the grouping by annotated class label because it is most intuitive.

5.1.3 Group-Averaging

The basic form of computing NAP values is to compute the average of the activations over a group of examples. Given the activations of a layer $A_l(X_i)$ for all $X_i \in \mathbf{G}$, the NAP of group $\mathbf{G}$ is computed as

$$NAP_{\mathbf{G}} = \frac{1}{|\mathbf{G}|} \sum_{X_i \in \mathbf{G}} A_l(X_i). \tag{5.2}$$

With this averaging, variation of the individual examples is removed and group-common information are retained. A NAP of a single group is of only little use as it misses the relation to other groups. Therefore, the more useful NAP is the set of group-NAPs according to a grouping.

$$\mathbf{NAP}_{\mathbb{G}} = \{NAP_{\mathbf{G}}\}_{\mathbf{G} \in \mathbb{G}} \tag{5.3}$$

5.1.4 Averaging with Normalization

While the averaging characterizes the activations over the groups of inputs, we are particularly interested in the differences between the activations. For example, there can be neurons that are highly activated by the inputs from each group. The activity of such neurons can overshadow other neurons that are less active but more class-distinctive. Therefore, to improve the class-specificity of the information, we propose to apply a normalization of the averaged activations $\mathbf{NAP}_{\mathbb{G}}$ (Equation (5.3)). As normalization approach, we compute the average activation over the entire data set and subtract this global average from each group NAP. The normalized $\mathbf{NAP}_{\mathbb{G}}$ cannot be interpreted as activations anymore but as the difference from the baseline activations. Positive values indicate a characteristically high neuron activation in comparison to the entire data set and negative values a comparably low neuron activation. With this approach, the activation differences between the groups are emphasized. The enhanced contrast is shown in an exemplary activity visualization of an MNIST MLP in Figure 5.2.

$$\begin{aligned} \text{normalizer:} \quad & \overline{A_l} = \frac{1}{|\mathbf{X}|} \sum_{X_i \in \mathbf{X}} A_l(X_i) \\ \text{normalized } \mathbf{NAP}_{\mathbb{G}} &= \left\{ NAP_{\mathbf{G}} - \overline{A_l} \right\}_{\mathbf{G} \in \mathbb{G}} \end{aligned} \tag{5.4}$$

In practice, computing the normalizer over the entire data set has two disadvantages. Firstly, the computation can be expensive due to the high number of examples. Secondly, for strongly imbalanced data sets, the normalizer can become close to the $NAP_{\mathbf{G}}$ of the most overrepresented group. This leads to normalized values that effectively are relative to one group instead of to the common baseline activations across the groups. To circumvent these problems, we propose another normalization approach which computes the normalizer as an average over the group NAPs.

$$\begin{aligned} \text{normalizer:} \quad & \overline{NAP_{\mathbf{G}}} = \frac{1}{|\mathbb{G}|} \sum_{\mathbf{G} \in \mathbb{G}} NAP_{\mathbf{G}} \\ \text{normalized } \mathbf{NAP}_{\mathbb{G}} &= \left\{ NAP_{\mathbf{G}} - \overline{NAP_{\mathbf{G}}} \right\}_{\mathbf{G} \in \mathbb{G}} \end{aligned} \tag{5.5}$$

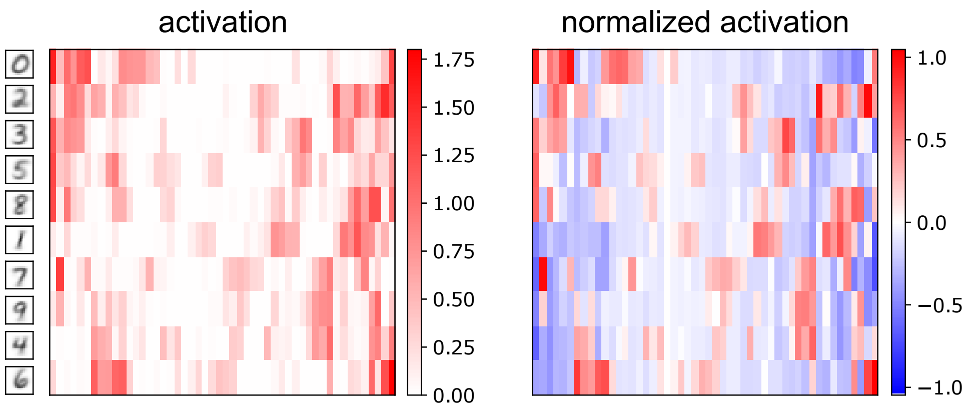


Figure 5.2: NAPs for a hidden layer in a MLP trained on the MNIST data set before and after normalization. Row and column orders are based on the group and neuron similarities, respectively (compare Section 5.1.8).

This type of normalization overemphasizes small groups in the baseline. However, the group comparability suffers less from this effect compared to one group dominating the baseline. For computational efficiency and to improve group comparability in the presence of overrepresented groups, we use the second normalization as the default normalization approach.

5.1.5 Handling Unaligned Data

In order to obtain useful averaging results, we require data that are aligned. If the features of interest in the input data are in different positions, the averaging removes both the variation between the inputs and the characteristic features. Unfortunately, data are rarely obtained such that they are readily aligned. Most data sets also come without detailed annotation that informs about properties with a high-resolution. Often, the only annotations are categories that the data fall into. For specific use cases, there exist registration techniques to align data to each other [46, 29, 78]. However, these alignment methods are not generally applicable to every type of data, in particular considering that we require to not only align the input data but also activations. Therefore, we propose a fast approach to estimate the position of important features.

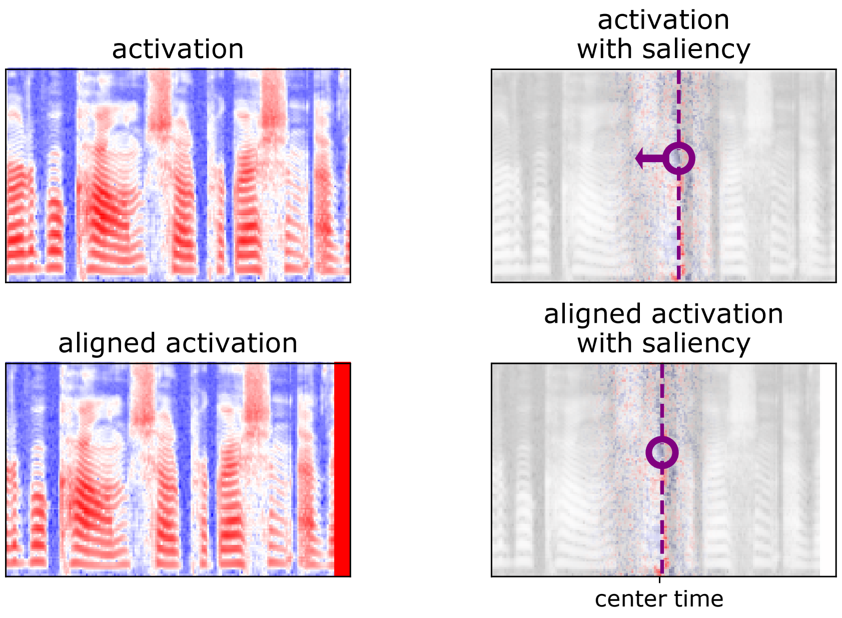
Our estimation of important features is based on how sensitive the model prediction is to them. We follow the assumption that an input or activation value is relevant for the prediction if its change affects the model prediction strongly, and that this relevance indicates the presence of a relevant feature.. These features are what we want to align our data on to move the group-specific information to a common alignment position. To obtain the prediction-relevance, we compute saliency information using sensitivity analysis as described in Section 3.5. Consider a data example X_i, its activations $A_l(X_i)$ in layer l and the logit of the highest active neuron $N_{pred} = A_{out}(X_i)\left[\arg\max(A_{out}(X_i))\right]$ in output layer out. We compute the saliency as

$$S_l(X_i) = \frac{\partial N_{pred}}{\partial A_l(X_i)}. \tag{5.6}$$

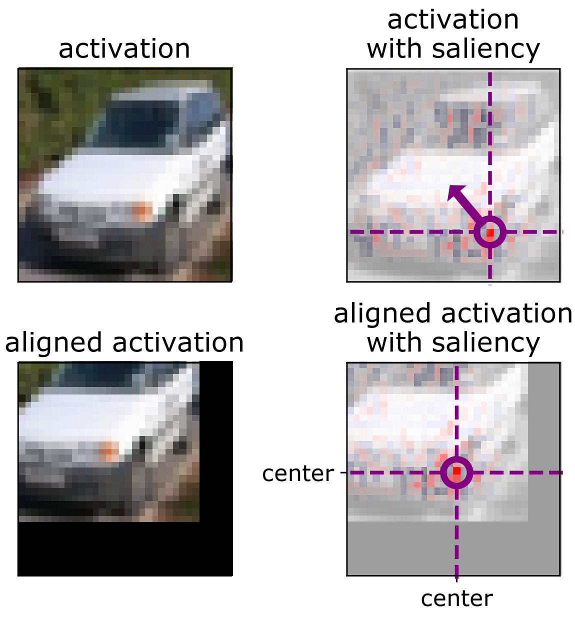
From the saliency values, we obtain the position of the highest value as the alignment point. Then, we perform the alignment of the corresponding activations such that, for each dimension, the most salient point is in the center of the activations. We achieve this by cropping values on one side of each dimension and padding with zeros on the opposite side. We apply the equivalent transformation to the saliency values to keep them aligned to the activations for later steps.

Figure 5.3 demonstrates the alignment step for two examples in the input layer. The alignment does not need to be performed for every dimension. For example, in audio data, it is more suitable to only align along the time dimension, while image data is more suitably aligned in both image dimensions. In the 1D-convolutional case (Figure 5.3a), we align the data in the time dimension only. The 2-dimensional activations of the image example (Figure 5.3b) are aligned in both image dimensions. Generally, alignments along the channel dimension are strongly discouraged because changing the location of neurons or feature maps leads to not being able to identify them in the resulting NAP.

Improvements Saliency values can be positive and negative indicating whether a decrease or increase of the activation value improves the prediction, respectively. As sensitivity in both directions of effect indicates a relevance, it can be helpful to use the saliency values as their absolute values $S_l(X_i) = |S_l(X_i)|$. This way, patterns that improve the prediction by decreasing their values can also be identified as features to align the activations on.



(a) Alignment in the time dimension.



(b) Alignment in both image dimensions.

Figure 5.3: Procedure of aligning activations such that they are centered at the point of highest prediction-relevance. For (a) and (b), respectively, the top row shows activations and superimposed saliency values before the alignment and the bottom row displays results of the alignment procedure. The purple circles indicate the respective position of highest absolute saliency value.

If there is a channel dimension in the saliency values, it is useful to average them along this dimension and obtain the most salient position from the channel-averaged values. This way, instead of finding a position which is most salient in any of the channels, the method identifies the position which is most prediction-relevant across all channels.

Because sensitivity analysis is known to produce high-frequency patterns, the saliencies $S_l(X_i)$ can be optionally smoothed with average pooling with a small kernel. The smoothed saliency values then indicate the average prediction-relevance of the respective value and its neighbors.

5.1.6 Masking Prediction-Irrelevant Information

Characterizing the network response through its activations is group-specific but independent of whether the activations are relevant for the network prediction. To improve NAPs such that they become prediction-relevant in addition to being group-specific, we propose to weight its values based on the saliencies. Specifically, we create group-averaged absolute saliencies corresponding to the NAP over a group. We then scale the values of the averaged saliencies to the range of $[0, 1]$ and multiply them with the NAP to obtain a prediction relevance-weighted NAP. As we weight down values with low saliency while keeping those which are important for the prediction, we refer to this procedure as masking prediction-irrelevant information. Figure 5.4 shows the effects of applying the mask for NAPs without and with normalization. Many neurons that initially show a high activation value are weighted down by the masking, which can be observed from an increased amount of neurons of white color. This, in turn, highlights the prediction-relevant neurons that are not weighted down and hence appear more active in comparison.

$$\begin{aligned}
\overline{S_{l\mathbf{G}}} &= \frac{1}{|\mathbf{G}|} \sum_{X_i \in \mathbf{G}} |S_l(X_i)| \\
\text{saliency mask: } M_{\mathbf{G}} &= \frac{\overline{S_{l\mathbf{G}}} - \min(\overline{S_{l\mathbf{G}}})}{\max(\overline{S_{l\mathbf{G}}}) - \min(\overline{S_{l\mathbf{G}}})} \\
\text{masked } \mathbf{NAP}_{\mathbb{G}} &= \{NAP_{\mathbf{G}} \cdot M_{\mathbf{G}}\}_{\mathbf{G} \in \mathbb{G}}
\end{aligned} \tag{5.7}$$

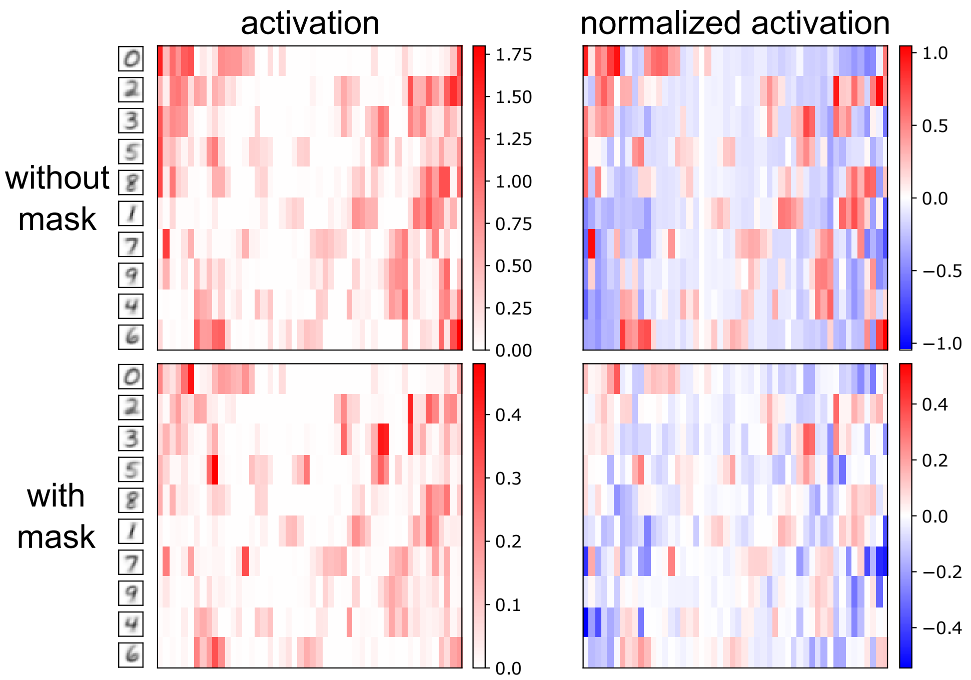


Figure 5.4: NAPs for a hidden layer in an MLP trained on the MNIST data set. Top: Activations averaged over the respective class without and with normalization. Bottom: Resulting NAPs when removing prediction-irrelevant activations based on saliency values. All plots use the same row and column orders, which are based on the group and neuron similarities, respectively (compare Section 5.1.8).

5.1.7 Subgrouping

Choosing useful groupings is crucial for performing analyses with NAPs. Here, we discuss different ways to adapt the aforementioned standard approach of using the annotated or predicted classes as grouping.

Annotation-Based and Manual Subgrouping

Groupings do not need to be based on the annotated classes. If available, additional annotations can be used to define groupings. For example, consider a data set which includes labels that are not used for the model task, but only exist as meta data. These labels can be used as a grouping as well. If there are no additional annotations, it is also possible to manually create them. As creating annotations for the entire data set involves huge amounts of manual work, this approach is better suitable for creating incomplete groupings to relate specific smaller groups of interest to each other, which we discuss in a following paragraph in this section.

Subgrouping by Prediction Errors

The groupings into annotated and predicted class naturally extend to subgroupings that can be used to investigate prediction errors made by the model. One possible subsetting is to use the grouping by the predicted class $\mathbb{G}_{pred}$ with $|\mathbf{C}|$ groups according to the classes $\mathbf{C}$, and divide each group into subsets of correctly and wrongly classified examples.

$$\begin{aligned}\mathbf{G}_{c+} &= \left\{X_i \in \mathbf{X} \text{ where } pred(X_i) = c = lbl(X_i)\right\}\\ \mathbf{G}_{c-} &= \left\{X_i \in \mathbf{X} \text{ where } pred(X_i) = c \neq lbl(X_i)\right\}\\ \mathbb{G}_{err,binary} &= \{\mathbf{G}_{c+}, \mathbf{G}_{c-}\}_{c\in\mathbf{C}}\end{aligned} \tag{5.8}$$

Further, this type of error subgrouping can be extended to every type of possible combination of predicted and annotated class. This creates groups that do not only represent whether an error was made but also which class the contained examples are misclassified as. Such grouping can intuitively be understood as creating one group for each position of a confusion matrix.

$$\begin{aligned}\mathbf{G}_{cp,cl} &= \left\{X_i \in \mathbf{X} \text{ where } pred(X_i) = cp \wedge lbl(X_i) = cl\right\}\\ \mathbb{G}_{err,pairs} &= \left\{\mathbf{G}_{cp,cl}\right\}_{cp\in\mathbf{C},cl\in\mathbf{C}}\end{aligned} \tag{5.9}$$

Automatic Subgrouping with Clustering

A group is most useful if it comprises examples that share common features. This is not necessarily the case for examples that are conceptually related. For example, consider the group of musical instrument images. There are various different types of musical instruments that cannot be characterized by a common feature. Computing a NAP for such a high-variance group will suffer from averaging out not only example variation but characteristic features, as well. In fact, many models categorize their inputs into conceptually broad categories. For example, consider a group of dog images without information about the breed of the dog. If there is no available annotation to automatically create fine-grained groups, automatic subgrouping can be performed. The idea of this approach is to use clustering algorithms to subdivide the high-level group into smaller (disjunct) subsets in which the examples are similar to each other and, hence, obtain subgroups with less variation. It is not trivial to obtain good subgroupings through clustering, though. Clustering in the input space is not representative because input value similarity does not properly represent conceptual similarity. A more concept-based approach is to use the activations in some deeper layer for the clustering. However, this relies on the model having learned meaningful concepts. Furthermore, it is not clear which layer yields the most useful information for clustering. Shallow layers only represent simple patterns and are no high-level concepts. Deep layers, on the other hand, might represent the data already similarly to the annotated classes such that clustering mainly distinguishes the classes instead of subconcepts.

Incomplete Groupings

In Equation (5.1), we defined groupings as a split of the entire data set. However, NAPs can be computed for sets of groups that do not encompass all data examples, as well. Using incomplete groupings has several use cases. Firstly, it can speed up the computation process. If there are classes with a lot of redundancy in terms of highly similar examples, only choosing a random subset of each group can be sufficient to estimate the NAP, while decreasing the computation time substantially. Secondly, it is useful to address specific questions about groups of interest. For example, consider a grouping in which a subset of groups shows similar activation patterns. NAPs based on a complete grouping capture the differences between the groups but they are overshadowed by the higher differences to other classes. Using only the similar groups for computing the NAPs emphasizes the differences between them. The group averages do not change by using only a subset of the groups, only the differences between them is emphasized due to computing the normalizer from the similar groups only. Therefore, this improved contrasting is only possible with using the normalization step. Lastly, incomplete groupings can also be useful to compare small subsets of examples with each other. This can help in answering highly-specific questions about the representational similarity of concepts defined by the user through manual selection of examples.

Overlapping Subgroups

Until now, we considered groupings to comprise disjunct sets of examples. However, NAPs can equivalently be computed for overlapping groups. This is particularly useful for data whose annotations follow a hierarchy. For example, one can investigate the group of dog images and the groups of images from the individual breeds at the same time. Overlapping groups are also useful, if data examples belong to multiple categories in multi-label classification tasks. The results of NAPs of overlapping groups are more likely similar to each other because they are computed from some identical examples. While this intuitively appears to be biasing the similarity analysis, if the examples really belong to the different groups the similarity of them is still a correct information.

5.1.8 Visualization and Similarity Analysis

Visualizing NAPs directly is only useful for layers whose activations are interpretable themselves. This particularly applies to input layers of models that process visually interpretable data. In Chapter 6, we address this issue and present topographic maps as a way to intuitively visualize the NAPs of any layer.

Here, we perform similarity analyses of NAPs with hierarchical clustering (see Section 2.1.2). We compute the pairwise Euclidean similarities between the NAPs. These can be visualized in a $|\mathbb{G}| \times |\mathbb{G}|$ heat map that represent the similarities as colors. Further, we apply hierarchical clustering with complete linkage to obtain a group hierarchy based on NAP similarities. Using the clustering information to sort the rows and columns of the heat map, we obtain a visualization as cluster map. A dendrogram which represents the clustering result is attached to both axes of the cluster map to better visualize the relations between the groups. An exemplary cluster map of MNIST classes based on the similarities of their NAP values in a CNN is shown in Figure 5.5.

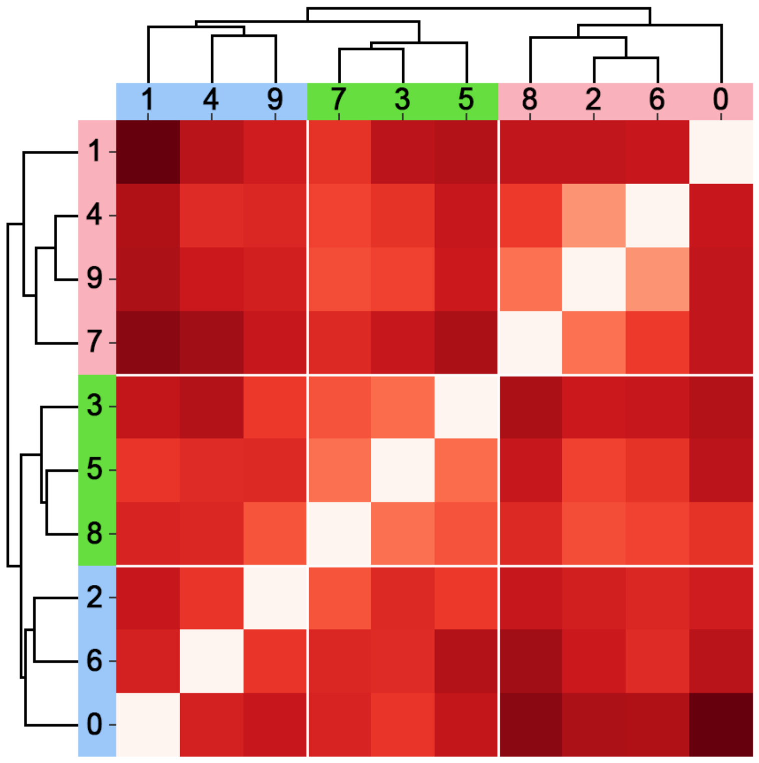


Figure 5.5: Visualizing relations of groups by creating a cluster map based on their activations for the classes of an MNIST CNN. Here, average activations over all examples from the same class are computed and their similarity is computed with a hierarchical clustering using Euclidean distance. White indicates identity and darker shades of red indicate less similarity.

For fully-connected layers or if using aggregated feature maps, we can create an overview of the NAP values. To this end, we visualize the NAP values of all groups $\mathbf{G} \in \mathbb{G}$ and neurons $N \in \mathbf{N}$ as a $|\mathbb{G}| \times |\mathbf{N}|$ heat map. Again, the neurons and groups can be sorted according to their similarity by a hierarchical clustering, respectively. Examples for visualizing NAPs of groups and neurons as heat map and as cluster map, that means, in original order and with ordering by similarity, are shown in Figure 5.6. While the heat map (top) appears scattered and is difficult to interpret, the cluster map (bottom) with rows and columns ordered by similarity allows to visually identify similar groups and neurons. For this visual comparison, it is not necessary that the values are interpretable themselves.

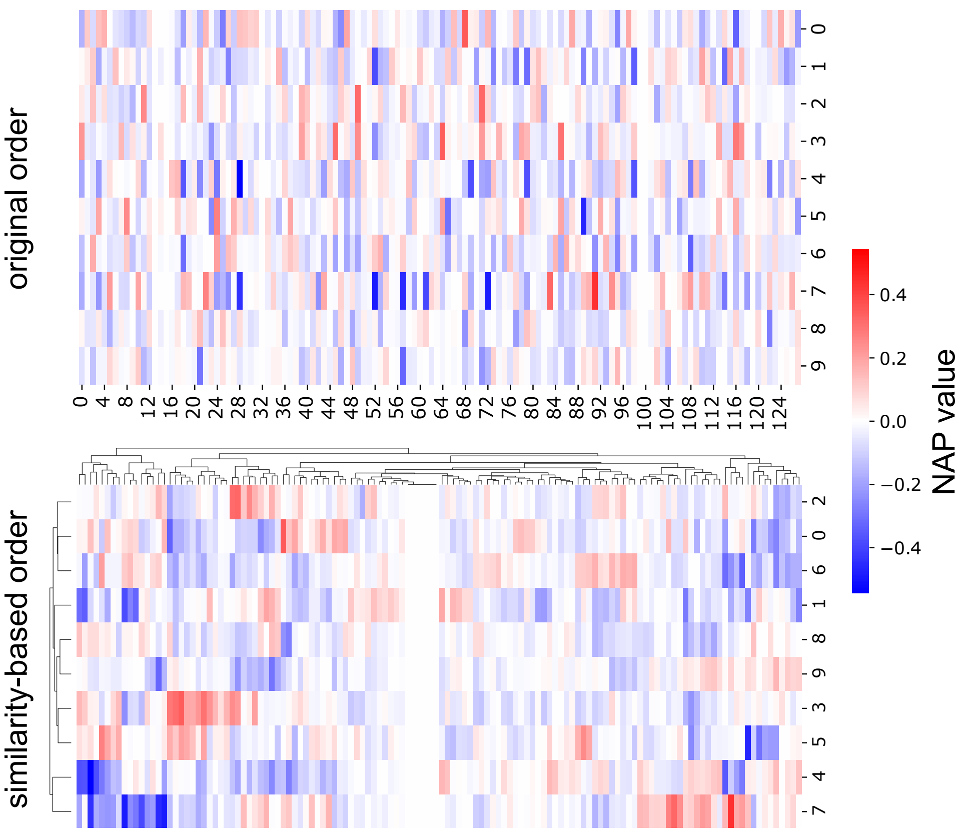


Figure 5.6: Heat map and cluster map visualization for normalized NAPs with applied saliency mask of the MNIST MLP model. Both plots show the same values but with a different ordering of the rows and columns. The cluster map allows to see similarity patterns between groups and neurons more easily.

5.1.9 Pipeline

Finally, we present in which order the described steps are applied when computing NAPs. Figure 5.7 shows the entire pipeline and refers to the individual sections that describe the step in more detail.

First, we start with the data set and the layers of interest as inputs. For each data example, we obtain the activations in the chosen layers. If alignment is desired or prediction-irrelevant information shall be masked, we also compute saliency values for each activation. If using alignment, we use the saliency values to align both the activations and saliencies. Before applying the next steps, we need the grouping of interest as additional input. This can be supplied directly from, for example, the annotated

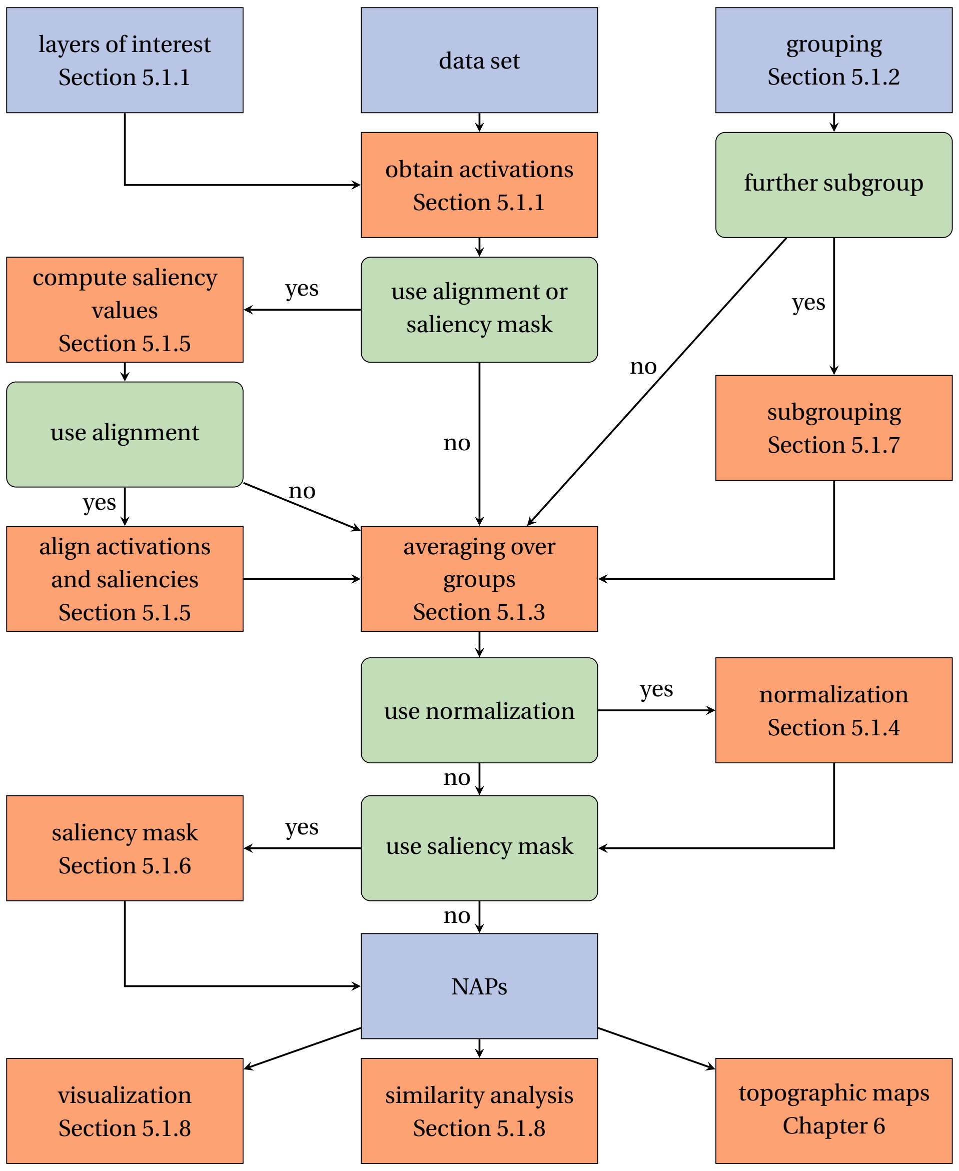


Figure 5.7: Computation pipeline for computing NAP and further use them to gain insights into the model. Inputs and outputs to the pipeline are represented in blue color, processing steps are orange and decisions about whether variations are used are highlighted in green.

class, or optionally from a further manual or automatic subgrouping. We then average the extracted (potentially aligned) activations of each of the desired groups. The averaged activations can then first be normalized. Next, the saliency mask for removing prediction-irrelevant information can be applied. Finally, we obtain the NAP for all groups in each layer of interest.

Following the path when deciding against each variation corresponds to the minimally required steps for computing NAPs: Obtaining activations and averaging them for each group. These steps are need to be applied in every other variation of the computation pipeline. The resulting NAPs can then be used in different ways to gain insight into the model. In particular, in this thesis, we demonstrate inspecting them directly, performing similarity analyses and visualizing them intuitively as topographic maps.

In Figure 5.8, we visually demonstrate the process of creating NAPs with alignment and masking prediction-irrelevant information for a hidden layer NAP of an exemplary ASR model.

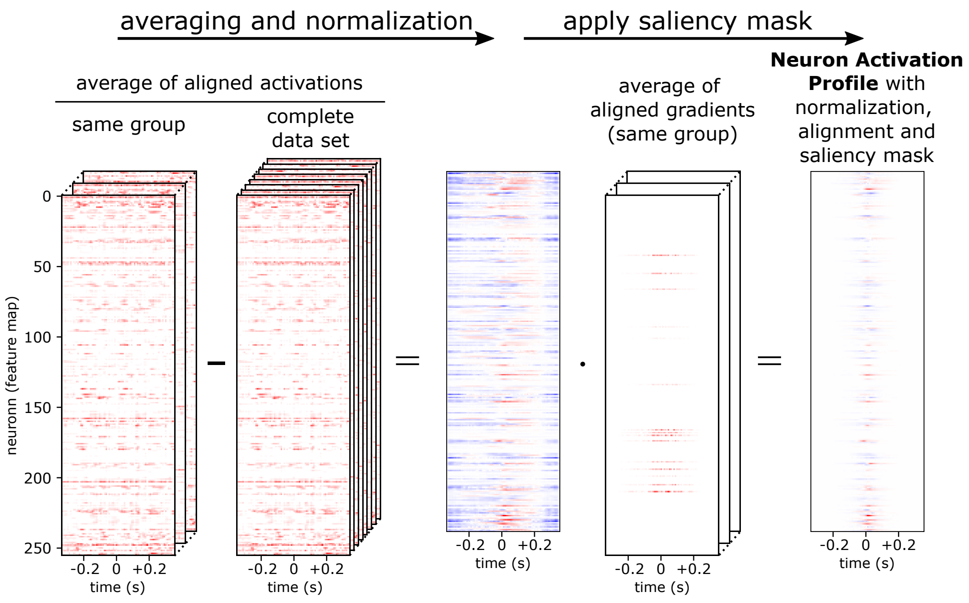

averaging and normalization
apply saliency mask
average of aligned activations
same group
complete data set
average of aligned gradients (same group)
Neuron Activation Profile with normalization, alignment and saliency mask
neuronn (feature map)
0
50
100
150
200
250
-0.2 0 +0.2
time (s)

Figure 5.8: Averaging with normalization and saliency masking to obtain a NAP suitable for speech data.

5.2 Averaging and Normalization

In this section, we investigate the effect of applying normalization to the averaging results. We expect this step to improve the interpretability of the activation values and to lead to a better representation for similarity analyses. We investigate the normalization for three cases. First, we demonstrate how the normalization affects the activation values and their distances for a two-dimensional toy example. Then, we will investigate the practical effect on a simple MLP and a 1D CNN.

5.2.1 Investigated Distance Metrics

In particular, we investigate Euclidean distance (Equation (5.10)) and Cosine distance (Equation (5.11)). We compute Cosine distance as 1− Cosine similarity.

For two n-dimensional data points p and q,

$$dist_{Euclidean} = \sqrt{\sum_{i=1}^{n} \left(p_i - q_i\right)^2} \tag{5.10}$$

and

$$dist_{Cosine} = 1 - \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2 \cdot \sum_{i=1}^{n} q_i^2}} \tag{5.11}$$

Euclidean distance can be any positive number while Cosine distance is in the range of $[0, 2]$.

5.2.2 Two-Dimensional Toy Example

For this example, we create a two-dimensional data set that comprises three data points. They represent the average activation of three imaginary groups and we manually assign them the two-dimensional values $[1.0, 1.2]$, $[2.0, 1.0]$, $[1.4, 2.0]$. Normalizing these values by subtracting the global average leads to the normalized values $[-0.47, -0.20]$, $[0.53, -0.40]$, $[-0.07, 0.60]$. Comparing the values before and after normalization, it is clear that the normalized values help to understand which values are small or large compared to the entire data set. Moreover, the normalization has beneficial effects on some distance metrics, as demonstrated in Figure 5.9.

The linear shift of all data points does not affect the Euclidean distance, hence they are identical for the original and normalized values. However, there is a beneficial effect on metrics that are based on angles between vectors, like the Cosine distance. Particularly for values that are non-negative,

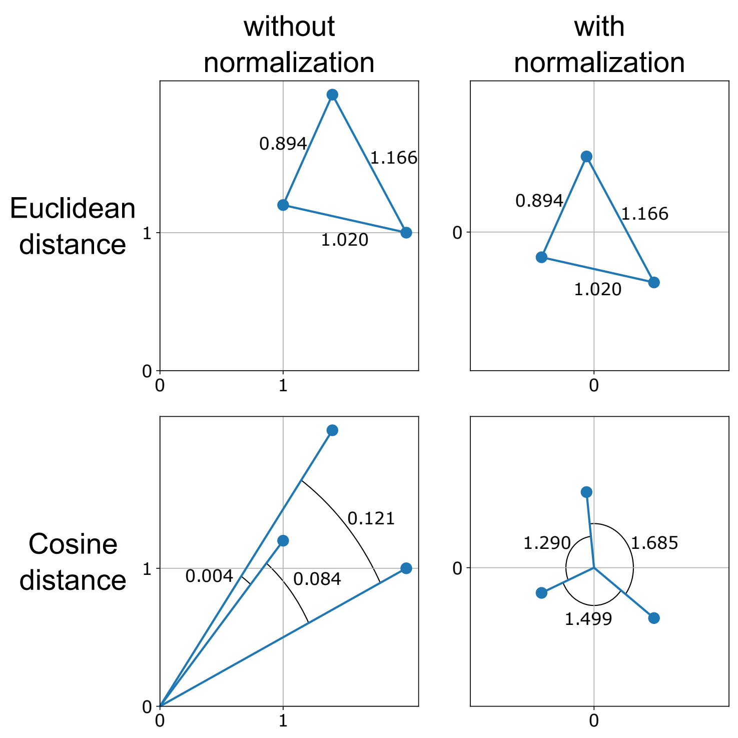


Figure 5.9: Effects of the normalization procedure on distance measures demonstrated for a two-dimensional toy example. Values on the edges and angles show the Euclidean and Cosine distances, respectively. Euclidean distances are not affected by the normalization but Cosine distances become better distinguishable.

for example ReLU activations, the possible value range of Cosine distances is restricted to $[0,1]$. By applying the normalization, the values are distributed around the center of origin and therefore, the full range of values $[0,2]$ is possible. In the toy example, we clearly observe that the Cosine distances between the data points are small and similar to each other for the original values, but are larger and better distinguishable after normalizing the data points.

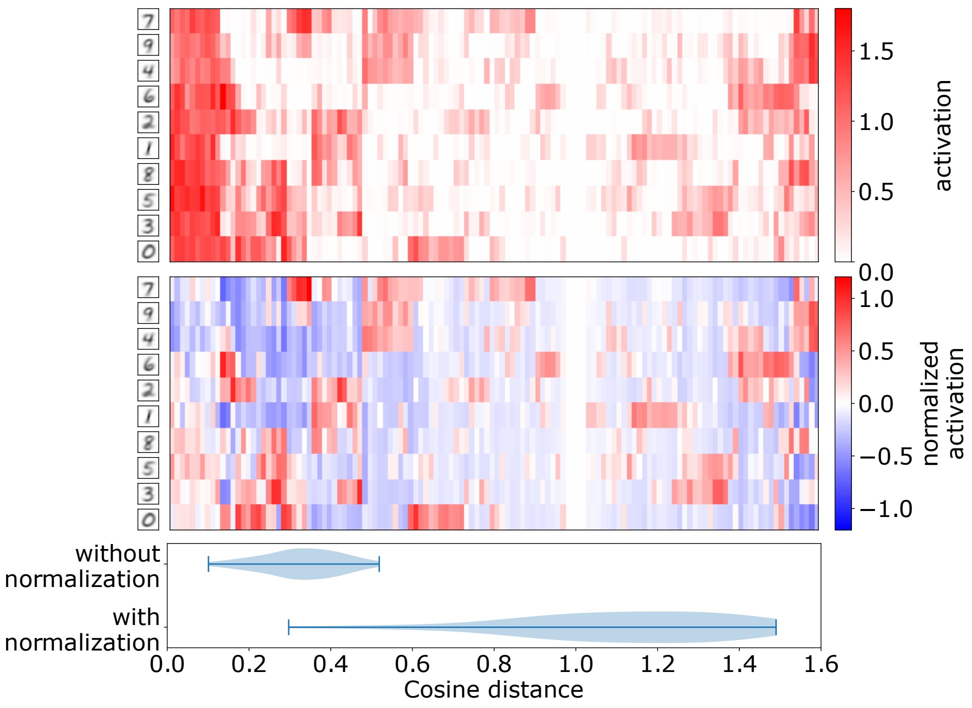
In fact, computing the Cosine distance for values that are normalized by subtracting their mean value in each dimension is equal to computing the Pearson Correlation distance:

$$\begin{aligned} dist_{Correlation} &= 1 - \frac{\sum_{i=1}^{n}(p_i - \overline{p}) \cdot (q_i - \overline{q})}{\sqrt{\sum_{i=1}^{n}(p_i - \overline{p})^2 \cdot \sum_{i=1}^{n}(q_i - \overline{q})^2}} \\ &= 1 - \frac{\sum_{i=1}^{n}(p_i - 0) \cdot (q_i - 0)}{\sqrt{\sum_{i=1}^{n}(p_i - 0)^2 \cdot \sum_{i=1}^{n}(q_i - 0)^2}} \\ &= 1 - \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2 \cdot \sum_{i=1}^{n} q_i^2}} = dist_{Cosine} \end{aligned}$$

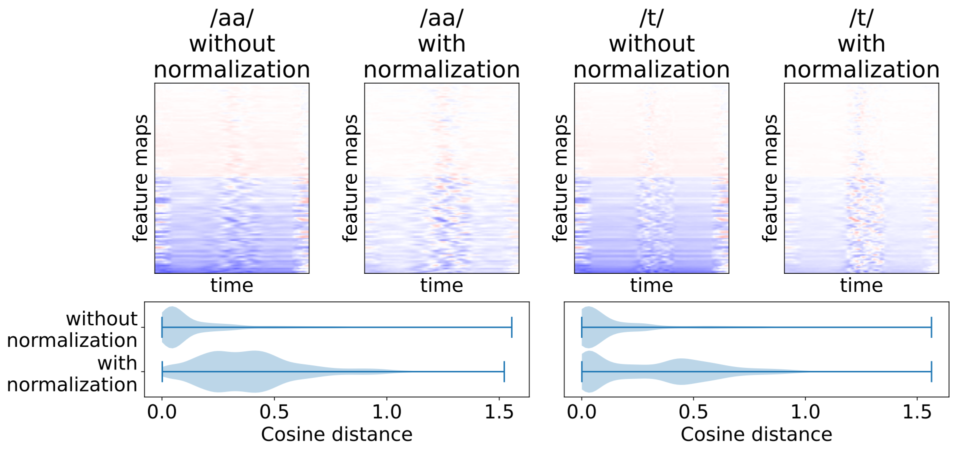
5.2.3 Hidden Layer NAPs

Figure 5.10a shows the activation averages for a hidden layer in a MLP trained on MNIST before (top) and after (middle) applying the normalization approach. We artificially altered some neuron activations to demonstrate the normalization effect. The left 20 neurons are highly active for all classes as can it be seen in the top subfigure. After the normalization, the activation differences of these neurons become more clear that they are comparably less active for classes ‘4’ and ‘9’ and more active for classes ‘3’ and ‘5’. The violin plot in the bottom of Figure 5.10a further shows the distribution of pairwise Cosine distance values between the groups. As in the previously shown toy examples, normalization leads to the Cosine distance values being spread out in a wider range of values making the groups easier to distinguish based on their distances.

Further, we demonstrate the effect for a hidden layer in a 1D-convolutional model trained on the TIMIT data set in Figure 5.10b. It shows the NAPs without and with normalization for two exemplary phonemes. The normalization leads to easier distinguishable feature maps between the groups. Moreover, it scales the activations such that feature maps which have similarly high or low activation across all groups do not appear as significantly active anymore.



(a) NAPs for a hidden layer in an MLP trained on the MNIST data set. Top: Activations averaged over the respective class. Bottom: Applying normalization by subtracting the global activation average. Both plots use the same row and column orders, which are based on the group and neuron similarities, respectively (compare Section 5.1.8).



(b) NAPs for a 1D-convolutional hidden layer in a CNN trained on the TIMIT data set for two exemplary phonemes. The feature maps are sorted by their average activation across the entire data set.

Figure 5.10: Activation normalization effects for an MLP and a CNN hidden layer.

5.3 Aligning Data

To evaluate the alignment step, we perform the following experiments. In the first experiment, we alter the MNIST data set such that the inputs are not aligned anymore and investigate whether our sensitivity-based alignment approach retrieves useful class-characteristic input patterns. After that, we continue to investigate the usefulness of the alignment for a convolutional ASR model as a more realistic application. First, we qualitatively evaluate whether the alignment improves the group-characterization for exemplary input layer NAPs. The, we perform an extensive quantitative evaluation by comparing the alignment results to ground truth annotations obtained from the TIMIT data set.

5.3.1 MNIST Variations

First, we demonstrate the efficacy of the alignment for a simple data set. To this end, we use shifted MNIST – a version of the MNIST data set in which we randomly position each 28 × 28 MNIST image in a 56 × 56 empty image. This way, we obtain a data set in which the images of the written digits are not aligned at the center of the image anymore (compare Section 4.1.1). Our alignment procedure is supposed to retrieve a useful alignment of the images again.

Figure 5.11 shows the obtained input layer NAPs with and without using the alignment step, both with only averaging the (aligned) inputs and with additionally applying the normalization. Without alignment (middle rows), we clearly observe that no useful information about the digit classes is characterized by the input layer NAPs, neither with averaging (grayscale) nor with normalized averaging (colored). This demonstrates how applying averaging unaligned data leads to severe information loss due to the different location of the objects in the data. After applying our saliency-based alignment procedure to all inputs, the input layer NAPs show much more class-characteristic patterns, as can be seen in the bottom rows of Figure 5.11. In the simple input averages that are shown in grayscale, we can already identify the typical shapes of many digits, for example, the round shape of ‘0’ or the characteristic single stroke of class ‘1’. Other classes are difficult to identify from the input averages, but become clearer when applying the normalization for achieving better contrast, as shown in the

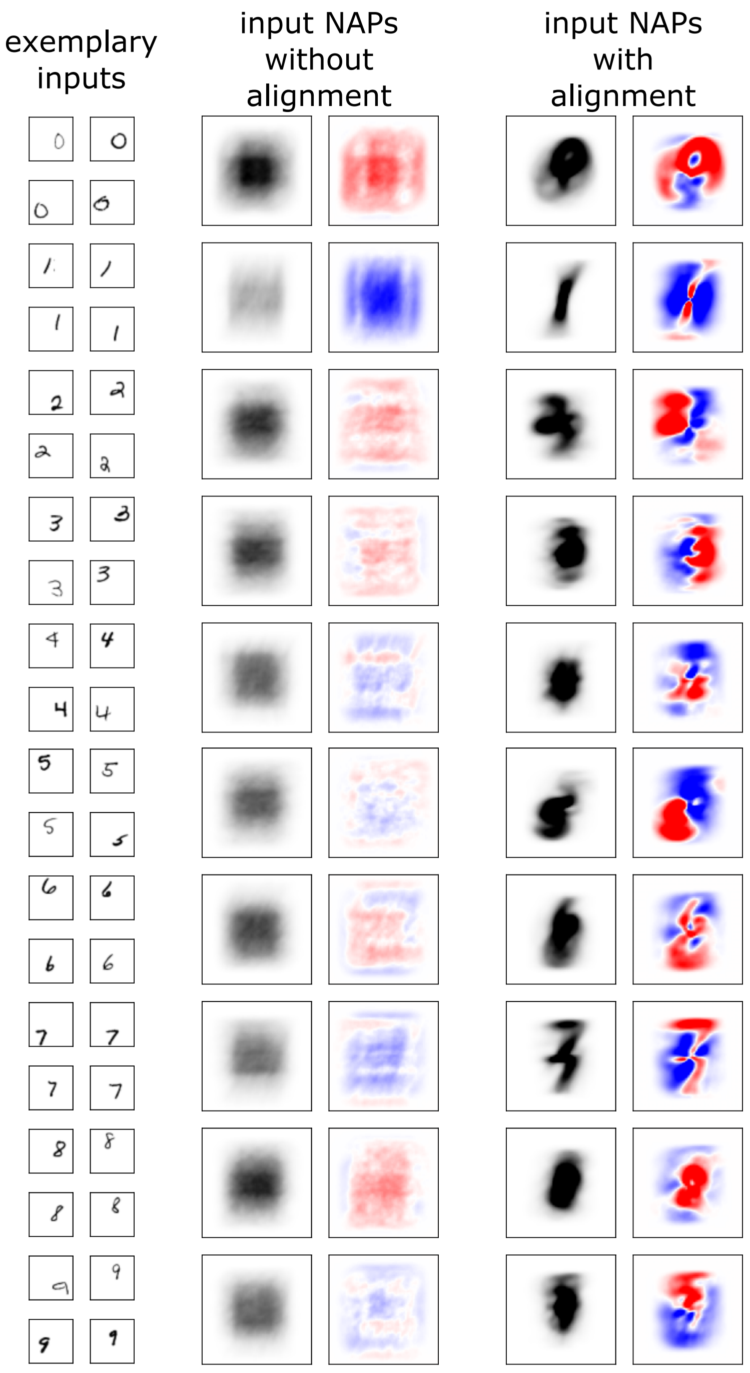


Figure 5.11: Input layer NAPs for a DNN trained on the shifted MNIST training data. NAPs are computed with and without normalization and alignment, respectively. Averaged inputs are shown in grayscale, using the original color range from pixel value 0 (white) to 255 (black). Normalized NAPs use a 0-symmetric color map from blue over white to red.

colored images. For example, the characteristic shapes of digits ‘3’ and ‘4’ are clearly visible in the normalized NAPs although the grayscale averages only show a round black spot. This is due to the higher contrast which emphasizes the gaps between the strokes as they have a comparably lower value. Note that the alignment is not always positioning the digits in the center of the images if the most prediction-relevant position is not in the center of the digits. Digit ‘2’, for example, is typically aligned at its bottom right. This is reasonable because the bottom horizontal stroke that ends at the bottom right is specific for this class and the network can use this feature to identify it. Further, we observe that the examples of some classes are not aligned to the same position for all examples. This is particularly clear for class ‘7’ where there appear to be two different alignment modes. Either they are aligned at the lower end of their diagonal stroke or at the horizontal stroke close to the junction of the two strokes. This leads to a pattern which mixes the typical shape of a 7 in the upper and the lower half of the NAP. Finally, digit ‘9’ is not recognizable from the NAP. One potential reason is that the DNN uses various different features to classify examples of this class due to a high variation of how the digits are written. We rather suspect that the network might have learned a strategy to only classify classes ‘0’–‘8’ based on their features and putting everything else in category ‘9’. In this case, it is reasonable that the alignment fails because the model does not use the actual class features for the prediction.

5.3.2 ASR – Qualitative Alignment Evaluation

While the MNIST variations are relatively simple data sets, we now investigate the alignment for the more complex TIMIT data and using alignment based on different ASR models. Again, as hidden layer NAPs are not visually interpretable, we use input layer NAPs to demonstrate how the alignment improves the description of the group-characteristic patterns in speech. Figure 5.12 shows exemplary input layer NAPs for the models W2L and W2P.

In the W2L model, we clearly observe that the alignment leads to better characterization of the input patterns for the letters ‘a’ and ‘t’. While without alignment both patterns almost look the same, after alignment we see clear differences. The vowel letter ‘a’ shows typical higher frequency intensity at

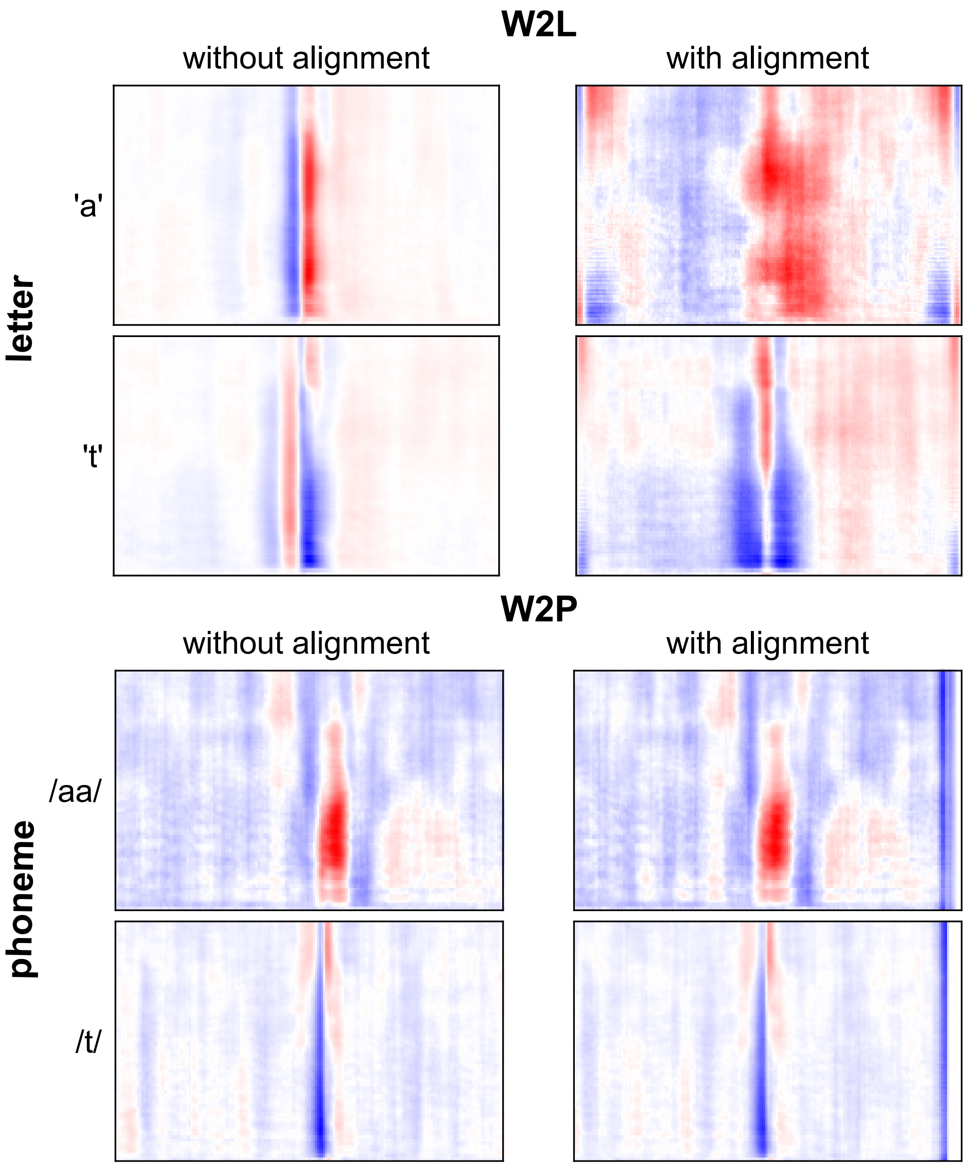


Figure 5.12: Input layers NAPs in models W2L and W2P with and without alignment for two exemplary phonemes and letters, respectively. Patterns in W2L become clearly more distinct through alignment while W2P patterns are only shifted. This indicates that our alignment technique is useful for data that are not aligned (W2L) as well as for data that are aligned but not at the center (W2P).

formants and the plosive letter ‘t’ has a high intensity of high frequencies and a fast change of power across all frequencies. However, the alignment does not always improve the patterns, as it can be seen in the patterns for the W2P model in Figure 5.12. The input layer NAPs for this model show a strongly pronounced padding on the right side of the NAPs, which indicates that the model only has a systematic shift of where it predicts the phonemes. Therefore, although the data is not aligned in the center, it is still aligned at some position. Hence, by applying the alignment, the resulting patterns do not change qualitatively but only regarding their location. It is not surprising that aligning already aligned data does not improve the results. However, we like to positively highlight that our alignment procedure does not un-align the data again.

5.3.3 ASR – Quantitative Alignment Evaluation

To further evaluate the alignment step of the NAP computation procedure, particularly for the case of speech data, we use the two models that predict phonemes: W2P and W2P_shallow. We expect our method to align at the time steps in the input spectrogram frame which are the actual occurrences of the predicted phonemes. Therefore, for each prediction, we additionally obtain the annotated phoneme at the center and at the alignment position. Moreover, we compute the time difference of the alignment position to the real occurrence of the predicted phoneme according to the annotation. We then use this distance to quantify the alignment error and refer to this measure as “alignment offset”. For instances that are predicted as a phoneme which is not contained in the annotation of the input spectrogram frame, we define the alignment offset to be -1. Because the alignment cannot be correct in these cases, the maximum possible alignment quality for our data set and both phoneme prediction models is to align 93.6 % of the instances correctly.

Alignment Quality – Model Average

W2P_shallow: Initially, only 3.7 % of the frames are annotated with the predicted phoneme in the center. Through alignment, we achieve 59.3 % of the frames for which the phoneme at the alignment position are equal to the annotation. Allowing an alignment offset of up to two time steps

(16 ms), the predicted phoneme is equal to the annotated one for 80.3 % of the frames. For the W2P_shallow model, we observe an average alignment offset of 26 ms.

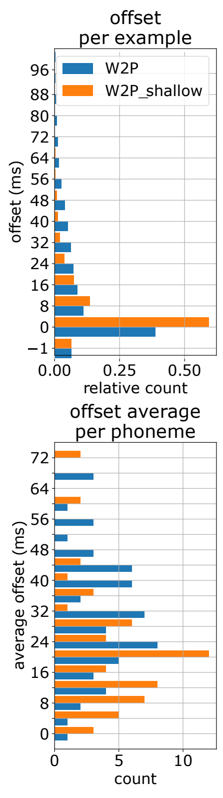
W2P: Only 3.6 % of the frames show matching prediction and annotation at the center time step initially. Alignment increases this number to 38.8 %. Considering also the annotated phonemes in a time frame around the alignment point, 58.6 % of the frames are correctly aligned. Corresponding to the smaller alignment accuracy compared to the W2P_shallow model, we also observe a higher average offset of 37 ms in the W2P model.

For the two phoneme prediction models, we conclude that our alignment method works as expected in a large number of frames. Because the averaging takes all instances into account, it is sufficient to align the majority of the frames correctly. Notably, the deeper model for phoneme prediction has a significantly worse alignment accuracy than the shallow model. We suspect that this is related to an overfitting of the deep model. An overfitted model does not learn meaningful features but memorizes information, such that it can perform correct predictions without focusing on the actual occurrence of the sound. Hence, for the W2P model, we argue that the alignment works properly despite the lower accuracy. The provided alignment accuracy metric for the phoneme models is computed as average over all phonemes, yet there are substantial differences between them. Therefore, we further investigate the alignment accuracy and offset for each individual phoneme.

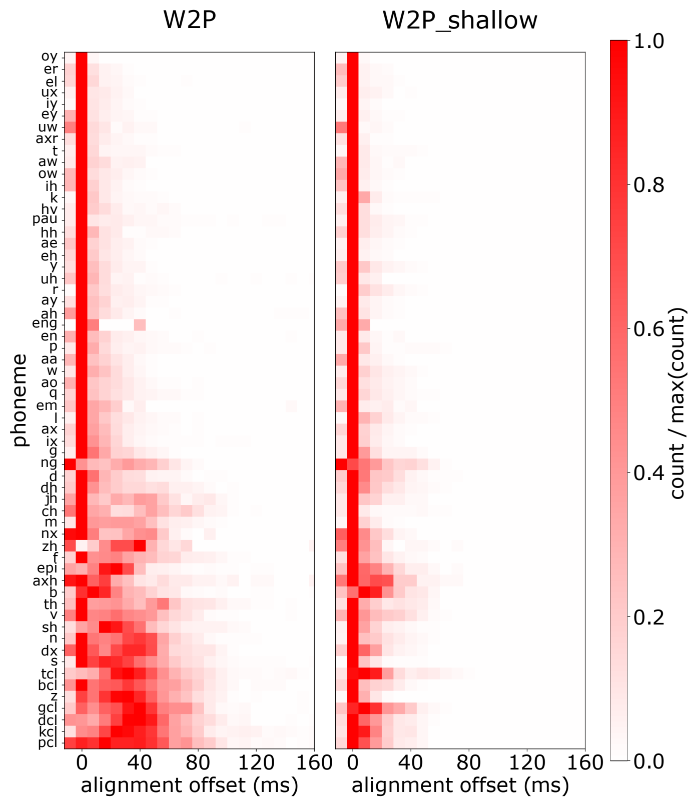
Alignment Quality – Per Phoneme

Figure 5.13a shows distributions offsets across all phonemes as an overview. In W2P_shallow, clearly more examples are aligned with no or only small offset. Correspondingly, we observe larger offsets for W2P than for W2P_shallow, supported by the higher average alignment offset of W2P.

Figure 5.13b shows the per-phoneme alignment offset distributions for the two investigated models. The rows are sorted by the average alignment offset of the corresponding phoneme in the W2P model. For W2P, 25 out of 60 phonemes are aligned without offset in the majority of instances. Six out of the seven phonemes of highest alignment offset are closure symbols (`pcl`, `kcl`, `dcl`, `gcl`, `bcl`, `tcl`), describing the closure of the respective plosives. For example, `pcl` represents the closure of `p`. Because the W2P model has a higher capacity and a larger receptive field than the W2P_shallow model, it might predict phonemes by finding correlations to other phonemes. Especially for the closure symbols, we intuitively expect that the model might facilitate that they always preceed the corresponding plosive. However, by manually inspecting the most frequent phonemes at the alignment point, we do not observe this behavior. Also, we so not find any other intuitively interpretable correlation which the model grasped.



(a) Offset distributions.



(b) Offset distribution heat maps for each phoneme.

Figure 5.13: Alignment offset overview in phoneme prediction models. Distribution of offsets for all examples ((a) top) and of the average alignment offset per phoneme ((a) bottom). (b) Offset distributions per phoneme in models W2P and W2P_shallow. Counts in the heat map plots are scaled to the range [0,1] for each phoneme, respectively. An offset value of −1 indicates that the predicted phoneme is not in the annotation of the example. The highest 5 % of the alignment offset values are excluded for plotting as their frequency is too low to be visible.

In the smaller W2P_shallow model, 44 phonemes are correctly aligned in the majority of instances. Also, a smaller alignment offset is observed for all closure symbols. More specifically, all closure symbols except for `gcl` and `tcl` are aligned without error in most of the instances. These observations support further that the decreased alignment performance for the W2P model is not attributed to our alignment technique but to the worse generalization capabilities of the model. Moreover, this contributes to considering the W2P_shallow model as being easier interpretable than the W2P model.

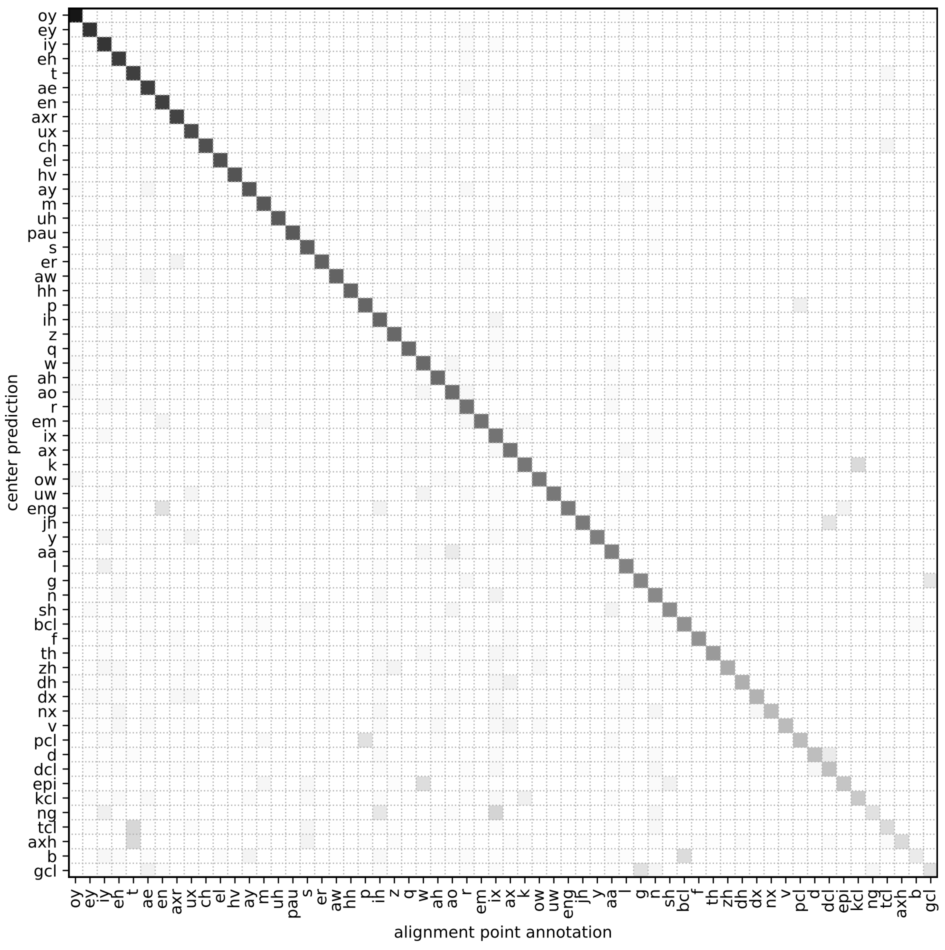


Figure 5.14: Alignment evaluation overview W2P_shallow. For each predicted class on the y-axis the relative frequency of the corresponding phoneme annotation on the x-axis after alignment is shown, with phonemes sorted by the maximum relative frequency. The color scale is [0,1] from white to black.

In addition to the alignment offset, we also investigate in more detail, which annotated phonemes the predicted phonemes are aligned to. In this section, we only provide a detailed plot for W2P_shallow because it generalizes better than the W2P model. Figure 5.14 shows the complete contingency table, for the predicted phoneme and the annotated phoneme after alignment. To get a better impression of how consistently which phonemes are aligned to a specific phoneme, we sort the table by the maximum value per row. We do not observe a clear tendency of a specific type of phoneme being aligned better than others. Moreover, the closure symbols are still aligned to various other phonemes, although the effect is not as pronounced as in W2P and vanishes when allowing a small alignment offset.

An overview of the evaluation plots for both models using alphabetical ordering of phonemes is provided in Appendix Figures A.1 and A.2.

Applicability to Letter Prediction Models

As discussed before, performing the same quantitative analysis for a letter prediction model is not possible. There is no unique mapping from letters to the phonemes, so we cannot test for equality of prediction and annotation. Moreover, a predicted letter can be correctly aligned to different phonemes, which leads to less specific alignment results than in the phoneme prediction models. Still, we can use the phoneme annotation to investigate which phonemes the predicted letters are aligned to. The contingency tables for W2L are shown in Appendix Figure A.3. Qualitatively, the alignment improves how well the annotated phoneme at the center position corresponds to the predicted letter. For example, without alignment, spectrogram frames that are predicted as letter `b` are very rarely annotated with the phonemes `b` or `bcl` at the center time step. After alignment, the majority of the frames predicted as letter `b` is centered at these phonemes.

5.3.4 Aligning Data – Summary

Using saliency maps based on sensitivity analysis is suitable for aligning data such that the relevant features of the individual examples are at a very similar locations. Therefore, our proposed alignment procedure enables us to average initially unaligned data. However, the sensitivity-based alignment is not a perfect substitute for an informed alignment of the data because it is dependent on the decision processes within the model. If the model uses different features to predict the object of interest, the alignment aligns the data at different features and consequently leads to inconsistent results in the averaging (like digit '9' in Figure 5.11). Typically, if there are only few different modes of features associated with the same prediction, there are still enough examples to obtain group-characteristic patterns. However, they then appear multiple times at different positions and their values are downweighted due to the averaging (like digit '7' in Figure 5.11). Moreover, the dependence on the model performance causes the alignment to be sensitive to the generalization of the model. If the model relies too strongly on patterns that are specific to individual examples, the alignment will align at them instead of at general group-characteristic features. This leads to higher loss of information when averaging data. In summary, we recommend to use pre-aligned data, either by design or by applying an informed alignment procedure, to obtain the most reliable results using NAPs. In cases where an informed alignment is not applicable, we suggest to use our proposed sensitivity-based alignment as it significantly improves the characterization of the network responses compared to simple averaging, especially for well-generalized models.

5.4 Improve Efficiency

Computing NAPs for entire data sets is computationally very expensive because it involves processing each individual example and computing averages over huge sets of high-dimensional activations. For large data sets and models, processing the entire data set is not computationally feasible. Therefore, we investigate how to improve the NAP computation efficiency by approximating the NAPs. In typical data sets for training DNNs, there are many similar examples that belong to the same group. However, small variations in the data do not affect the resulting NAP of this group due to the averaging. Therefore, approximating NAPs is possible by taking random subsets of the data. Randomly drawing examples from the data can miss out rare variants of particular groups, however, omitting them does not affect the NAP computation. Due to averaging over groups, activity related to rare examples is already underrepresented in the full NAP.

In this section, we investigate whether using random subsets of each group are sufficient to characterize the group-specific responses of the DNN. First, we perform a qualitative evaluation by visually inspecting input layer NAPs for different subset sizes and comparing the results between pre-aligned, unaligned and sensitivity-aligned data. Secondly, we quantify the precision and robustness of the approximations for subsets of different sizes according to their difference to the original NAP and how well the group-similarities are retained. Finally, we investigate whether the number of examples within a group influences how large the random subset needs to be to obtain a good NAP approximation. To this end, we compare full and approximated NAPs using relative and absolute subset sizes for a data set with highly unbalanced class sizes.

5.4.1 Visual Inspection of Approximated NAPs

We first visually inspect approximated input layer NAPs using different subset sizes. To investigate whether the approximation quality is dependent on the variation within the group, we compare the classes of padded and shifted MNIST data sets. For the latter, we further compare the NAPs with or without sensitivity-based alignment. The padded MNIST data set has only little variation within the groups because all examples are aligned. Only the image size is enlarged by padding to fit the image size of the shifted MNIST data set. In contrast, the shifted MNIST has higher variation as the digits are positioned randomly in the larger image. Computing the NAPs using alignment is expected to reduce the variation introduced by the random positioning. However, we do not expect it to achieve the low variation of the padded MNIST data set. Corresponding to the variation within the groups, we expect that the padded MNIST NAPs can be approximated with fewer examples than shifted MNIST. Moreover, with using aligned activations for computing NAPs for shifted MNIST, we expect to need fewer examples to approximate the original NAP than without alignment.

The MNIST training data set has 60,000 examples, with on average 6,000 examples per class. The number of examples is similar between the classes, ranging from 5,421 to 6,742. Due to the good class balance and the high similarity of the instances within each class, we use the same subset sizes for each of the classes. For comparability with a later experiment, we use relative values to define subset sizes. We investigate different subset sizes from using only a single random example to using all examples. Because increasing the subset size by a fixed number of examples has a larger effect for smaller subsets, we increase the investigated subset sizes exponentially. Specifically, we use subset size 1 as well as 10^k % for $k \in [-1, -0.75, ..., 1.5, 1.75]$, leading to 12 exponentially increasing subset sizes from 0.1 % to 56.2 %. Applied to the average amount of 6,000 examples per class, this corresponds to 1 to 3,374 examples as subset sizes. Further, we use 100 % as the reference, which, with 10^2 % examples, also represents the next value in the series of exponentially increasing subset sizes.

Figure 5.15 shows input layer NAPs for class '0' of the respective MNIST data set using different subset sizes using one randomly drawn set of inputs per subset size. The 100 % NAPs are the references that use all examples from the class. From visual inspection, it is clear that a very small random subset is necessary to approximate the reference NAP for padded MNIST. From around using only 1 % of the examples, the approximated NAP is visually not distinguishable from the reference. In contrast, approximated input layer NAPs for shifted MNIST without aligning the data need significantly larger subset sizes to be visually similar to the reference. Only the largest investigated subset size of 56.2 % yields a visually similar NAP. When computing the shifted MNIST NAPs with alignment, the required subset size to obtain visually well-approximated NAPs becomes smaller. The shown NAP with 1.8 % of the examples is already highly similar to the reference and from 10 %, it is visually not distinguishable anymore. This confirms our expectation that the required subset size for approximating a NAP depends on how much variation is in the group. Consequently, the required subset sizes also differ between the classes, for example, approximating class '2' NAPs requires more examples because the class has higher intra-class variance than '0'. Input layer NAP approximations for all ten MNIST classes are shown in Appendix Figures A.4 and A.5.

5.4.2 Quality and Robustness of Approximated NAPs

Using the same MNIST data sets and random subset sizes as in the previous subsection, we perform a quantitative evaluation of the quality and robustness of the approximated NAPs across all classes. First, we compute

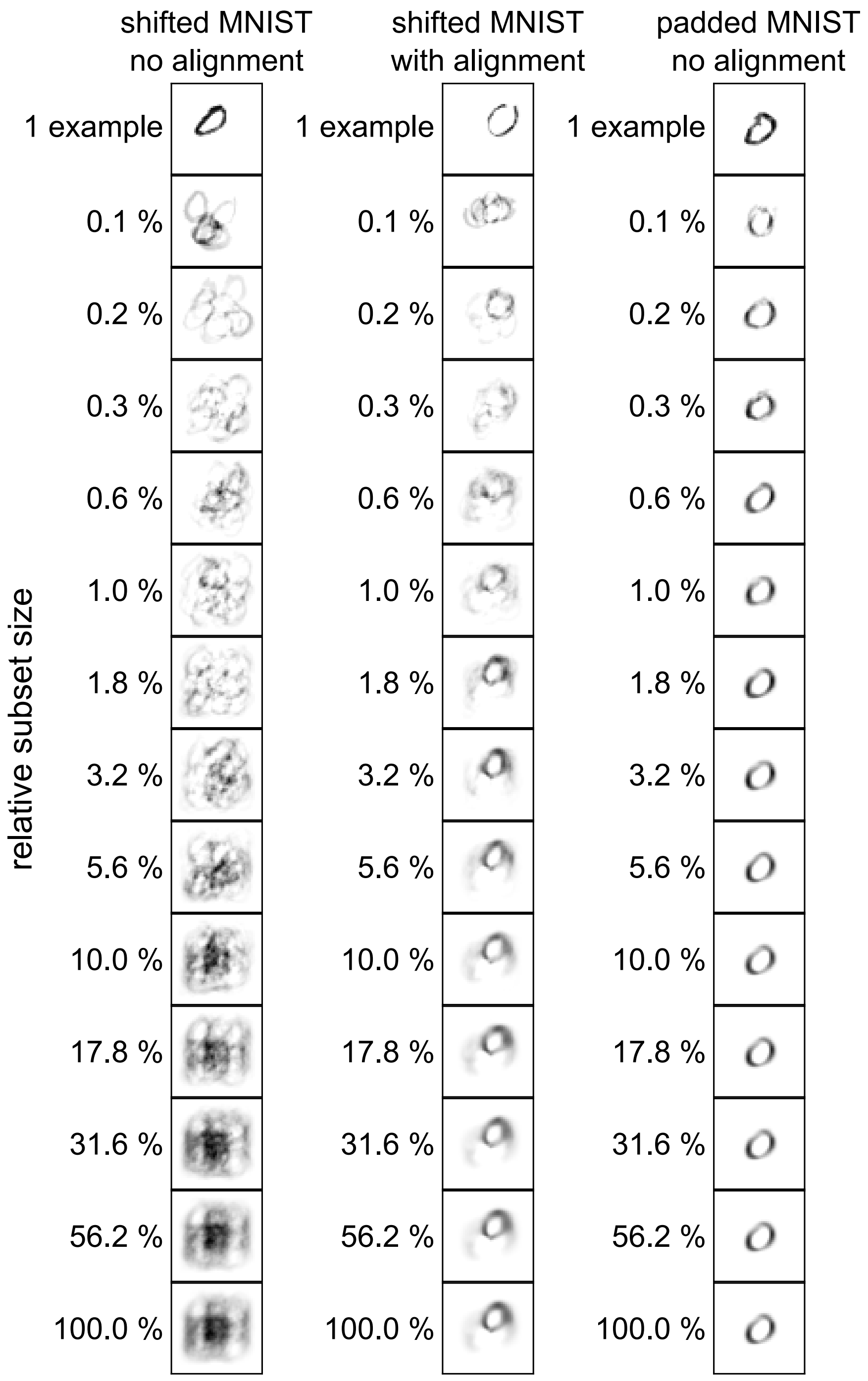


Figure 5.15: Approximated input layer NAPs using random subsets of different sizes relative to the number of examples of the respective group. Here, the exemplary class '0' is shown. All groups are provided in Appendix Figures A.4 and A.5.

NAPs using 100 random subsets of each investigated size. For each of them, we compute the mean absolute difference to the respective reference NAP and average the value across the ten classes. In addition, we investigate how the group similarities are retained when using approximated NAPs. To this end, we obtain reference group distance matrices by computing pairwise Euclidean and Cosine distances between the reference NAPs. We then compute the mean absolute difference between the group distance matrices using the random subsets with the reference distances.

The distributions of differences between NAPs and and between group similarities and their respective approximations for the random subset sizes for the input layer is shown in Figure 5.16. In general, larger subset sizes yield better approximations and show less variation between the random samples, which is not surprising. We again consistently observe that the approximation quality is best for the NAPs for the pre-aligned padded MNIST data and worst for the shifted MNIST data set. Still, the aligned shifted MNIST data has better approximation quality at the same subset size compared to not aligning the shifted MNIST examples. The absolute and relative differences show the same qualitative results but shifted MNIST without alignment has a substantially larger approximation error in relation to the values of the 100 % NAP or its group distances.

Notably, in Figure 5.15, shifted MNIST without alignment appears to need clearly more examples to approximate the reference NAPs. However, the quantitative evaluation does not show this large discrepancy to the aligned NAPs. This is likely due to the different color scales that we used in the input layer NAP plots. We scaled the colors such that white is the smallest value and black is the largest value across all classes in each respective configuration (data, alignment and subset size). A common color scale for all three experiments would have caused the NAPs for shifted MNIST without alignment to be visually almost white because of the strong averaging out effect. This means, that the actual values of the NAPs of shifted MNIST without alignment have comparably smaller values which in turn also decreases the mean absolute differences of the approximations to the reference.

The effect of subset size on the Euclidean group distance differences is similar to the differences between NAP values themselves, both with respect to how well the differences are retained and with respect to the relation

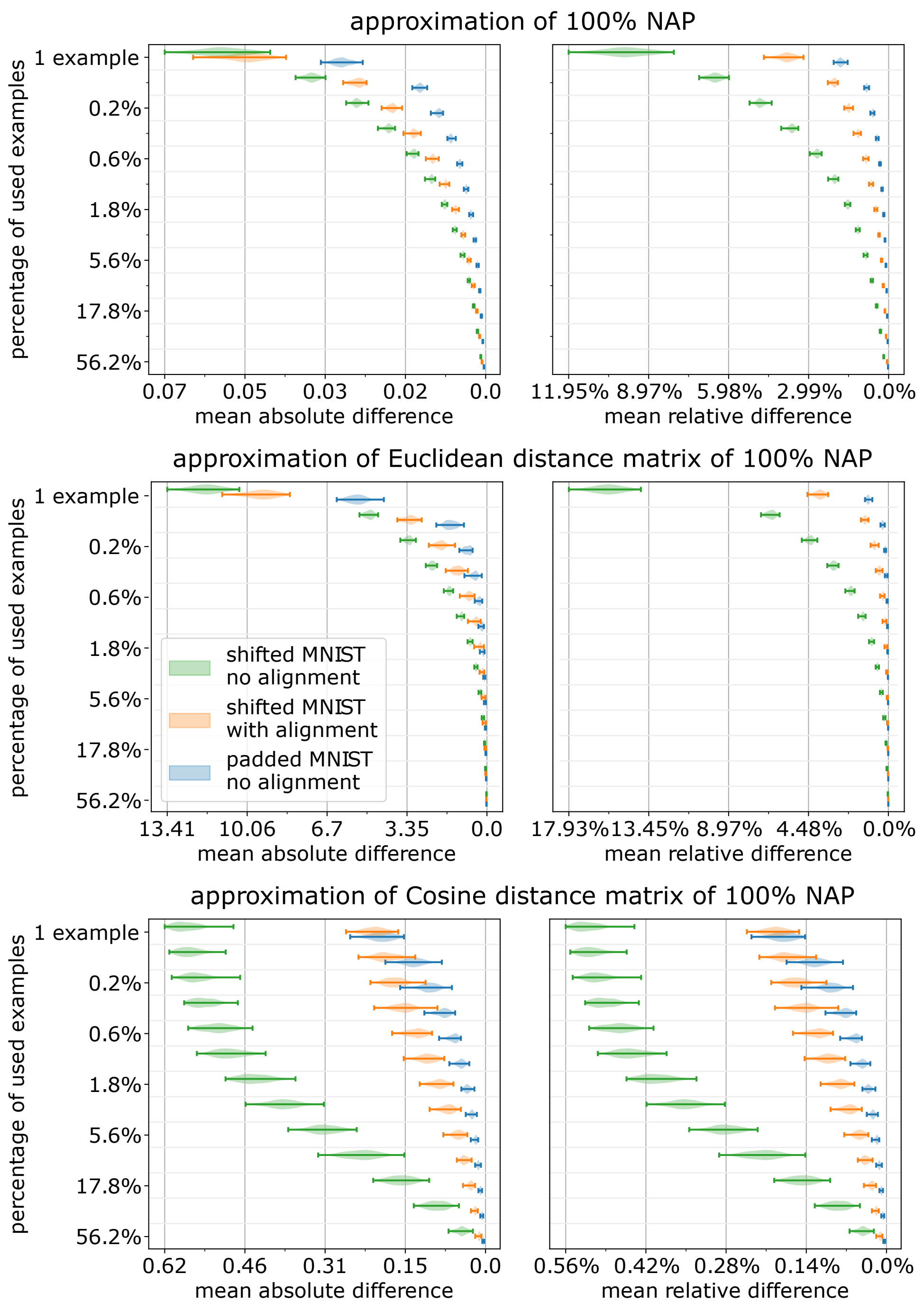


Figure 5.16: Difference of approximated NAPs to the NAPs computed on the entire data set. Differences are computed between the values (top) and the group distance matrices using Euclidean (middle) and Cosine (bottom) distance, as absolute (left) and relative (right) values. Distributions are computed over 100 different random subsets per investigated subset size.

between the different data sets. However, for Cosine group distance differences, we observe a distinct pattern. The approximation quality for shifted MNIST with alignment and padded MNIST behaves similar to the other measures, only with less improvement between smaller subset sizes and a slower convergence. Shifted MNIST without alignment, in contrast, needs significantly more examples to approximate the Cosine group distances adequately and small subset sizes do not substantially improve the approximation. This distinct pattern especially for shifted MNIST without alignment is likely due to the bad representativeness of the NAPs because of the high variation in the groups. Each NAP is highly similar, except for the magnitude of the values, leading to very small and similar Cosine distances between the groups. Euclidean distance, on the other hand, better distinguishes the classes based on the magnitude of the values and hence represents group distances that are closer related to the NAP value differences.

In the Appendix Figures A.6 to A.8, we show the approximation error plots for the three convolutional layers of the CNNs trained on shifted and padded MNIST. In deeper layers, the approximation errors become more similar comparing the different MNIST and alignment variants according to the absolute difference values. The relative values, however, show more similar patterns to those in the input layer.

In summary, we observe that it is possible to approximate NAPs with random subsets of the data. As expected, the higher the variation within the group, the fewer examples are necessary to obtain well approximated NAPs. This raises the question which metric is suitable to estimate in advance how large a subset needs to be in order to obtain a good approximation. One straight-forward approach is to explicitly investigate whether there is a relation between the variance within a group and the necessary subset size. However, the variance is also related to the number of examples in the groups. Therefore, in the following section, we investigate both the influence of group size and variance on different absolute and relative subset sizes.

5.4.3 Relation of Class Size and Variance with Subset Size

In this section, we investigate whether relative or absolute subset sizes are better suitable for approximating NAPs using TIMIT, which has an unbalanced size of the groups regarding the number of examples for different phonemes.

Experimental Setup

We compute normalized NAPs with alignment and saliency-masking based on the W2P_shallow model.

As relative subset sizes, we use the same values as in the previous experiments on MNIST. That means, we use subset size 1 as well as 10^k % for $k \in [-1, -0.75, ..., 1.5, 1.75]$, leading to 12 exponentially increasing subset sizes from 0.1 % to 56.2 %. Further, we use 100 % as the reference. The absolute subset sizes follow an exponential increase ranging from 1 to 12,302, oriented at the largest group which contains 23,556 examples. If a group comprises fewer examples than the subset size, we use the complete group.

We consider both the number of examples within the groups and their variation as potential contributing factors to how many examples are necessary to approximate the NAP of the respective group. To obtain the variance within a group, we compute the the median variance of each input value across the examples.

TIMIT contains 61 phoneme classes of different sizes of which we investigate three sets of phonemes in this experiment. Two sets contain four phonemes that are of similar size, respectively. The first set comprises phonemes between 1,238 and 1,384 examples and the groups in the second set are substantially larger phoneme classes with between 5,330 and 5,376 examples. With these two sets, we investigate whether similarly sized groups are similar in how many examples are necessary to approximate their NAPs and whether this is influenced by the intra-group variance. The third set of phonemes contains four groups of different sizes ranging from the smallest to the largest group and we use it to investigate how the group size influences the required subset size for approximating the NAPs.

Results

Figure 5.17 shows the NAP approximation qualities as the mean absolute NAP value differences to the reference NAPs for the three subsets (top to bottom), either using absolute (left) or relative (right) subset sizes.

Using absolute subset sizes, the approximated NAPs become almost identical to the reference when the subset size is at least as large as the size of the group, which results in the curves becoming vertical at this point. Small differences to the reference still occur due to normalizing by the global average of all random subsets. Apart from that, we observe only small differences of the approximation quality compared between the phonemes of any num-

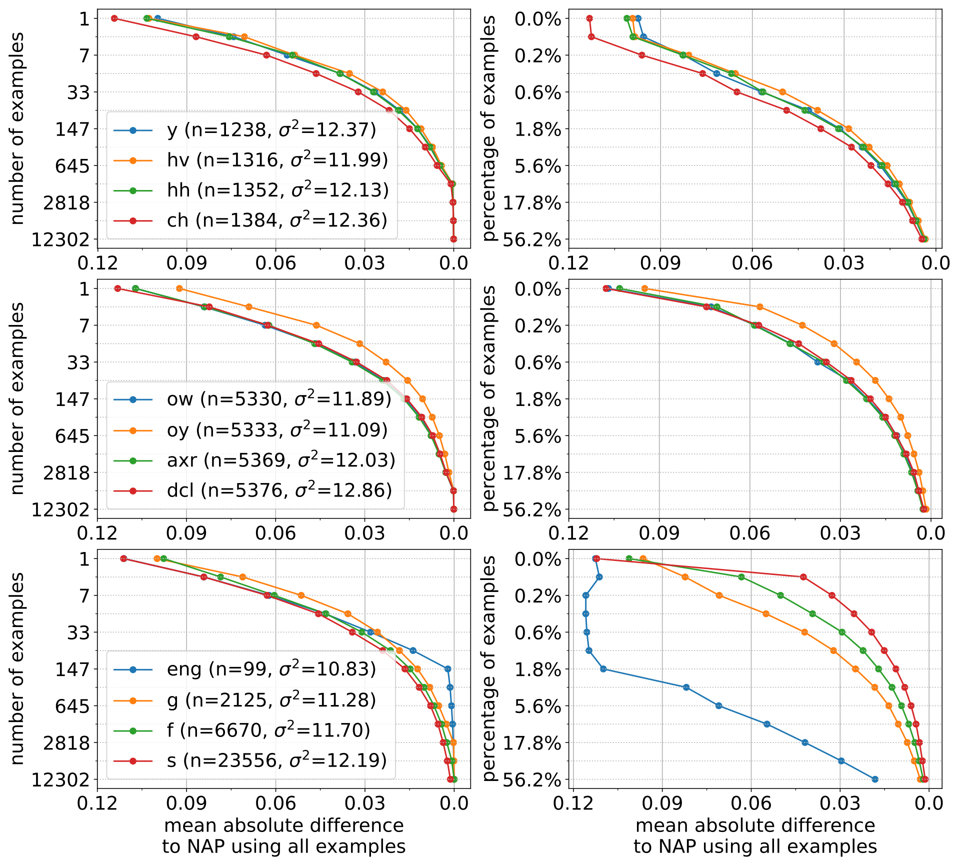


Figure 5.17: Difference of approximated NAPs to the NAP computed on the entire data set. Measurements are average values over 100 different random subsets per investigated subset size. Legends show the phoneme class, number of examples within this class (n) and the median value of the input values variances across the examples of the respective class (σ^2). Absolute and relative subset sizes are shown left and right, respectively. Top and middle rows show sets of classes of similar numbers of examples and the bottom row shows classes of different size, ranging from the smallest to the largest class. Note that approximated NAP of small groups like eng have slight difference to the reference for the complete group due to normalizing by the global average of all random subsets.

ber of examples. It seems that smaller groups can be approximated with smaller subsets, however, this pattern is not clearly pronounced. Moreover, we do not observe a relation between the variance within a group and how many examples it requires for approximating its NAP. The low-variance oy (middle left, orange) obtains better approximated NAPs with fewer examples than others of similar size. However, other phonemes of small variance, for example, eng (bottom left, blue), need more examples to obtain good approximations. There are also phonemes of higher variance, like y (top left, blue), which do not require more examples for good approximations.

For relative subset sizes, we observe an ordering by the class sizes both for large group size difference (bottom right) and similarly sized groups (top and middle right), although it is less clear for the latter. Larger groups mostly need smaller fractions of their examples to approximate the reference NAP. This is in agreement with the absolute subset sizes. As all groups need a similar number of examples to approximate their reference NAP, these subsets correspond to larger relative subsets for groups with fewer examples. However, there are exceptions like oy (middle right, orange) which needs relatively smaller subsets than most larger groups.

Our experiments show that the group size does not substantially affect how many examples are needed to approximate a NAP. This indicates that the averaging has an intrinsic upper bound where including more examples in the averaging does not change the average anymore. Hence, our results suggest that a fixed subset size is most suitable for computing approximated NAPs. Similarly, the intra-group variance appears to not affect the necessary subset sizes, however, for TIMIT, these variances are very similar to each other between the groups. This observation might not be reproducible in data sets with more diverse groups in terms of their variance.

5.5 Chapter Summary

This chapter introduced NAPs as a way to characterize DNN responses to groups of inputs in particular layers of the network. We presented our method and evaluated its individual steps.

Characterizing groups by averaging their activations is a suitable approach under several conditions. The first condition is that the data need to be aligned such that either the objects or the relevant features are at the same location in all inputs. Our approach works best for data that is pre-aligned or can be aligned by provided ground-truth annotations. If this is not possible, our sensitivity-based alignment procedure provides a convincing means to align the data such that the averaging yields group-characteristic patterns. As second condition, there should not be too large intra-group variance as this blurs the averages too strongly. In particular, the groups need to be really related in terms of which features characterize them to obtain the most useful results.

Normalizing the averages by subtracting the global average is recommended because it increases the contrast and improves the comparability between the groups, especially when using Cosine distance. For visually interpretable data, however, normalization can be detrimental in the input layer, as input layer NAPs are more intuitive to interpret in the same value range as the original data.

While computing NAPs over the entire data set is computationally expensive, they can be approximated by using random subsets. We empirically found that the same subset size can be used for groups of different size, but the optimal subset size is dependent on the data set and how exact the NAP approximation needs to be.

An in-depth exemplary application of using NAPs to interpret a DNN for ASR is provided in Section 7.2.

6 Topographic Activation Maps

In this chapter, we address our second research aim (RA2, Section 1.1) of creating visualizations of the NAPs to visually compare activity of a DNN between the different groups. To this end, we introduce topographic activation maps which are inspired by how brain activity is commonly displayed.

The methodology and experiments presented in this chapter are based on one of our publications [92]. In this thesis, we provide more detailed explanations and extend the evaluation of the results.

The chapter begins with a description of the methodology, followed by experiments to evaluate our novel visualization technique qualitatively and quantitatively. Application examples are provided in the following Chapters 7 and 8.

6.1 Method

In this section, we describe our proposed pipeline to compute the topographic activation maps. This includes obtaining the hidden layer representations of groups of examples, computing the layout of the topographic maps and visualizing the activations according to the layout. A visual abstract of the pipeline is shown in Figure 6.1.

6.1.1 Quantify Activation

For the layout computation, we use NAPs (Chapter 5) to characterize the activations for each group. Figure 6.1A summarizes computing normalized $\mathbf{NAP}_{\mathbb{G}}$ for two groups (compare Section 5.1.4), but without saliency align-

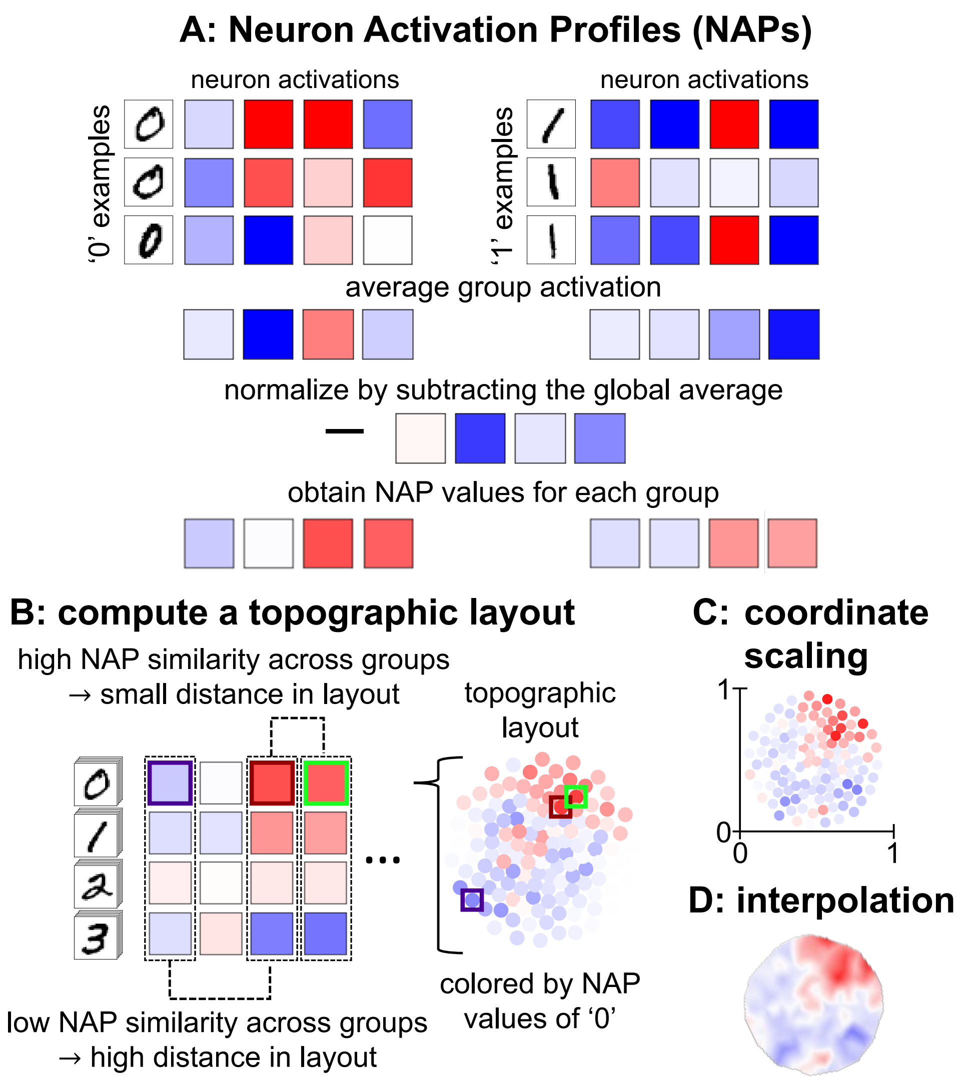

A: Neuron Activation Profiles (NAPs)
neuron activations
'0' examples
'1' examples
average group activation
normalize by subtracting the global average
obtain NAP values for each group
B: compute a topographic layout
high NAP similarity across groups → small distance in layout
topographic layout
colored by NAP values of '0'
low NAP similarity across groups → high distance in layout
C: coordinate scaling
1
0
0
1
D: interpolation

Figure 6.1: Visual summary of the computation of topographic activation maps. A: Obtain NAPs to characterize the DNN activations for the groups of interest, represented by two exemplary MNIST classes '0' and '1'. As described in Chapter 5 the average activation of each group is computed and normalized by the average activation of all examples to obtain a NAP, in the figure without alignment and masking. B: Compute a layout in which similarly activated neurons are in the vicinity of each other. The layout can be considered a dimensionality reduction of the NAPs to two dimensions. Then, the layout can be colored by the NAP values of the respective group. C: Scale the coordinates in both dimensions to a range from 0 to 1 for comparability of different layouts. D: Apply interpolation to achieve a continuous coloring which is visually more appealing than the scatter plot and resembles the neuroscience inspiration better.

ment or masking. However, in some experiments in this chapter, we also use normalized NAPs with saliency-alignment (compare Section 5.1.5).

6.1.2 Topographic Map Layout

To compute the layout of the topographic maps, we distribute the neurons of a hidden layer in a two-dimensional space. In general, we aim to compute a layout in which neurons of similar activity are in the vicinity of each other (Figure 6.1B). While using the same layout for all groups, we aim for activation similarity of nearby neurons for each individual group. We explore different techniques to compute such layout and describe them in the following paragraphs.

Self-Organizing Map (SOM)

We investigate SOMs because they are neural networks in which the neurons are arranged in a two-dimensional layout and trained such that neighbors are similar to each other [81] (compare Section 2.1.2). That means, SOMs are designed to obtain a topographic layout of their neurons. We use the MiniSom package [67] for Python to compute a SOM layout of the neurons based on their NAP values. For a layer with neurons $\mathbf{N}$, we compute a square SOM with shape $d \times d$ with $d = \lfloor \sqrt{|\mathbf{N}|} + 1 \rfloor$, such that there can potentially be one SOM position per DNN neuron. We train the SOM for 10 epochs with the NAPs as training data and use the default parameters of the MiniSom package. Then, we assign each neuron the coordinate of the SOM position whose weights are most similar to the NAP values of the neuron. However, multiple neurons can match best to the same position in the trained SOM and hence are indistinguishable in the layout. Therefore, for each set of neurons that share the same coordinate, we distribute the neurons uniformly on a circle centered at the coordinate assigned to the neurons. To ensure that the redistributed neurons are still closer to each other than to other neurons, considering that the SOM coordinates are integer-valued, we choose a circle radius of 0.2.

Co-Activation Graph

Further, we use a layouting method that considers the most similar pairs of neurons but not their exact similarity values. To this end, we use a co-activation graph approach which follows the idea of layouting a graph structure in which nodes and edges represent neurons and their activation similarity, respectively. We first compute the pairwise Cosine similarity of neurons according to their NAP values. We then create a graph with one node corresponding to each neuron. For each of the 7.5 % most similar pairs of neurons, we draw an edge between the corresponding nodes in the graph. We empirically choose the distance threshold based on the connectedness of the graphs for several MLP models and layers. The resulting graph can have separated subsets of nodes, which leads to large gaps in the layout. To avoid these gaps, we further ensure that the entire graph is connected. To this end, we first identify all maximal subsets of nodes where a path exists between all nodes, called connected components. Then, we link each smaller connected component to the largest one by drawing an edge between the most similar pair of neurons out of the two components. Finally, we layout the connected graph with the force-directed Fruchterman Reingold algorithm [40] using the NetworkX package [159]. From the obtained graph, we use the node coordinates as the topographic layout of the neurons. For brevity, we refer to the co-activation graph technique as the "graph" method.

Dimensionality Reduction

We test the popular dimensionality reduction methods PCA [60], tSNE [185] and UMAP [116] (compare Section 2.3) to project the high-dimensional neuron activation data into a lower-dimensional space while preserving most information.

Principal Component Analysis (PCA) [146, 60, 66] is a traditional unsupervised technique for linear dimensionality reduction. We use the first two principal components of a PCA obtained using the decomposition module of Scikit-learn (sklearn) [148]. UMAP [116] and tSNE [185] are more recent non-linear dimensionality reduction algorithms. We use the tSNE implementation from the sklearn manifold module. For stability of the tSNE algorithm, we initialize the learned embedding with PCA. To create two-dimensional UMAP projections, we use the python module umap-learn [115].

Particle Swarm Optimization (PSO)

For our PSO approach (compare Section 2.1.4), we initialize a swarm of particles in a two-dimensional solution space, where each particle corresponds to a neuron. With the PSO, we aim to optimize two aspects that we encourage with designated update rules. First, similarly active neurons shall be in the vicinity of each other. Second, the neurons shall be evenly distributed in the layout without gaps or dense clusters of neurons. The second objective makes the PSO approach different than the aforementioned approaches which focus on optimizing the layout based on activation similarity.

To achieve activation similarity of neighboring particles, we introduce a global force which is computed based on the actual NAP values. The global force encourages particles of similar neurons to attract each other while activation dissimilarity repels the corresponding particles: For the set of particles P, we compute a force for each particle $i \in P$:

$$\begin{aligned}
f_{glob}(i) &= \frac{1}{|P \setminus i|} \cdot \sum_{j \in P \setminus i} \left(attr_{glob}(i,j) - rep_{glob}(i,j) \right) \\
attr_{glob}(i,j) &= a \cdot \left(1 - \frac{dist_{NAP}(i,j)}{\max(dist_{NAP})^3} \right) \\
rep_{glob}(i,j) &= b \cdot e^{-\frac{1}{c} \cdot dist_{NAP}(i,j)}
\end{aligned} \tag{6.1}$$

In Equation (6.1), f_{glob} = global force, $attr_{glob}$ = global attraction, rep_{glob} = global repulsion, $dist_{NAP}$ = Cosine distance matrix of the NAPs. For the global force, we set the weight parameters to $a = 1.5, b = 0.5, c = 2$.

To obtain a well-distributed layout, we use a local force that only depends on the particle coordinates. Like the global force, it consists of an attraction and a repulsion term. However, in the local force, attraction closes gaps in the layout by penalizing large distances between pairs of particles and repulsion avoids that two particles occupy the same position. For the set of particles P, we compute a force for each particle $i \in P$:

$$\begin{aligned}
f_{loc}(i) &= \frac{1}{|P \setminus i|} \cdot \sum_{j \in P \setminus i} \left(attr_{loc}(i,j) - rep_{loc}(i,j) \right) \\
attr_{loc}(i,j) &= a \cdot \left(dist_{coord}(i,j) + 1 \right)^{-3} \\
rep_{loc}(i,j) &= b \cdot e^{-\frac{1}{c} \cdot dist_{coord}(i,j)}
\end{aligned} \tag{6.2}$$

In Equation (6.2), f_{loc} = local force, $attr_{loc}$ = local attraction, rep_{loc} = local repulsion, $dist_{coord}$ = pairwise Euclidean distances between particle coordinates. The values that we use for the weight parameters of the local force are $a = 1.5, b = 15, c = 2$.

We optimize the PSO for $T = 1000$ steps by updating the coordinates according to the weighted average of global and local force (Equation (6.3)). In early steps t, we use a high global force weight w_g to encourage the activation similarity of neighboring particles and then gradually increase the local force weight w_l to better distribute the particles in the space.

$$\begin{gathered} f = \frac{w_g \cdot f_{glob} + w_l \cdot f_{loc}}{2} \\ w_l(t) = \frac{1}{2} \cdot \left(\frac{e^{s(t)} - e^{-s(t)}}{e^{s(t)} + e^{-s(t)}} + 1 \right) = 1 - w_g(t) \quad \text{with } s(t) = \frac{9 \cdot t}{1000} - 3 \end{gathered} \tag{6.3}$$

PSO with Non-Random Initialization

The PSO method with random initialization needs careful balancing of the weight parameters of global and local attraction and repulsion. To require less fine-tuning of parameters, we investigate a variant of the PSO. Instead of optimizing the activation similarity with the global force, we compute an initial similarity-based layout with one of the aforementioned methods. We then only use the local force to further optimize the resulting layout with PSO. As the local force is independent of the activation similarities, the PSO is only used to equally distribute the neurons in the two-dimensional space. We use the same parameters as for the PSO method with random initialization except for setting $w_g = 0$ (Equation (6.3)) in all optimization steps. Using either the UMAP, TSNE, graph, SOM or PCA method to initialize the PSO, we call the hybrid methods UMAP_PSO, TSNE_PSO, graph_PSO, SOM_PSO and PCA_PSO.

6.1.3 Visualization

Finally, we use the NAP values (Section 6.1.1) and the layout coordinates (Section 6.1.2) to create topographic map images. To be able to compare different layouts, we first scale the layout coordinates such that in both dimensions the minimum value is 0 and the maximum value is 1 (Figure 6.1C). Then, we assign each layout coordinate a color according to the NAP value of the corresponding neuron for one group. We choose this color by mapping the NAP values to a symmetric continuous color scale, where

$$\begin{aligned} -\text{absmax}(\mathbf{NAP}_{\mathbb{G}}) &\quad \text{is blue} \\ 0 &\quad \text{is white} \\ +\text{absmax}(\mathbf{NAP}_{\mathbb{G}}) &\quad \text{is red,} \\ \text{with} \quad \text{absmax}(\mathbf{NAP}_{\mathbb{G}}) = \max\big(-\min(\mathbf{NAP}_{\mathbb{G}}), \max(\mathbf{NAP}_{\mathbb{G}})\big) &\quad . \end{aligned}$$

Then, we linearly interpolate the colors between the points with a resolution of 100×100 px (Figure 6.1D). We use the same interpolation resolution for all methods because of the applied coordinate scaling. Equal colors in topographic maps of different groups represent the same NAP value, but the colors can correspond to different values in each experiment or layer.

Colors of CNN Feature Maps For CNNs, we use the complete feature map NAPs to compute the topographic map layouts but we need to aggregate the activation values per feature map to obtain a color for the topographic map. We investigate four different aggregation techniques in our experiments.

Commonly used aggregation techniques are global average pooling and global max pooling. The average value indicates how well the filter that corresponds to the feature map matches across the entire input. It can show both high and low activity but highly specific patterns that are detected in only small regions might not be reflected well. Global max pooling identifies the activity at the position where the input is most similar to the feature represented by the respective filter. Therefore, it is expected to reflect activity of highly-specific filters better than global average pooling. However, pattern mismatches (negative activations) are not considered and the range of expected values is strongly biased towards highly positive values which might lead to worse distinguishable feature maps with respect to their aggregated value. We investigate these two aggregation techniques, referring to them as “mean” and “max”, respectively.

Specific to our NAP technique, we investigate the center value as an additional aggregation. When applying the saliency-based alignment (see Section 5.1.5), we relocate the most relevant position in a feature map to the center position. Therefore, we expect that using the NAP value at this position can be a good representation of the activation of a feature map. We refer to the center value aggregation as “center”.

In addition, as a random baseline, we investigate a random aggregation technique. To this end, we use the value from each feature map at a common randomly drawn position to obtain a color value. For the quality computation, we perform 10 random aggregations, of which we compute the average quality. The aggregation method “random” refers to this average value.

6.2 Experimental Plan

6.2.1 Evaluation Measures

Qualitative Criteria

Our technique aims to provide a comparative visual overview of the representations of groups in a DNN. The topographic maps are supposed to be easy to visually compare and they shall be perceived as similar to topographic maps in neuroscience. We consider the topographic maps to be visually similar to their neuroscientific inspiration if they have a round shape, contain no regions without neurons and show distinguishable regions in all groups. To achieve comparability between topographic maps, they shall have a similar shape and size and they need to be discriminable for dissimilar groups. We qualitatively evaluate these expectations by visual inspection.

Quantitative Evaluation Measures

In addition to the visually inspecting the topographic mas, we quantify their visual quality. To this end, we investigate different measures which aim to quantify aspects of the aforementioned visual quality criteria.

Perturbation-Based Measures To test whether each position in a topographic map is similar to its neighborhood, we apply a Gaussian blur to the topographic map image and compute the MSE between the original and the blurred image. However, this metric penalizes boundaries between regions strongly, but clear boundaries can be beneficial to distinguish different regions from each other. Therefore, we use a second metric based on image resizing with bicubic interpolation because it preserves edges better than using Gaussian blur. For this metric, we downscale the images, upscale them to the original size again and compute the MSE between the upscaled and the original image.

To not bias the quality metric based on the choice of a specific blur or resize strength, we compute the quality with different parameters for the the radius of the Gaussian blur and the downscaling size. We use Gaussian blur with ten different radii, ranging from 2 px to 20 px in steps of 2 px and investigate ten different downscaling sizes for the original topomap image of size 100 × 100 px from 55 × 55 px down to 10 × 10 px in steps of 5 px (see an example in Figure 6.2A).

To aggregate the results for the different parameter choices while interpolating the values for parameters in between, we use an estimated area under the curve (AUC) value. Using the parameters in the order of increasing effect of image alteration, we consider the resulting MSE values as function values and apply the trapezoidal rule with width 1 to estimate the AUC.

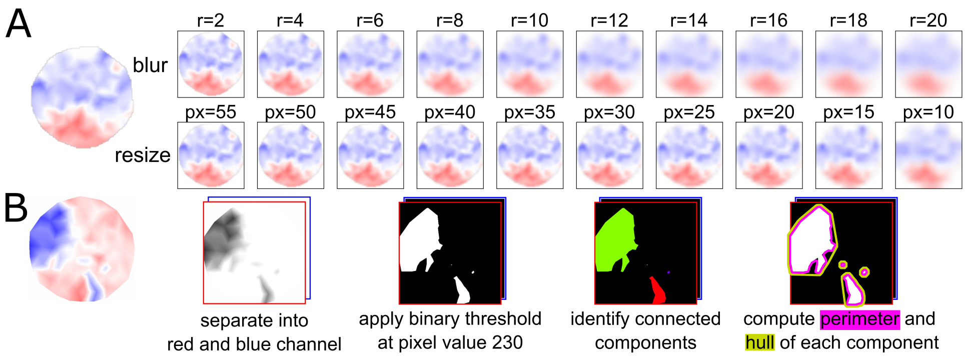

A
blur
resize
r=2
r=4
r=6
r=8
r=10
r=12
r=14
r=16
r=18
r=20
px=55
px=50
px=45
px=40
px=35
px=30
px=25
px=20
px=15
px=10
B
separate into red and blue channel
apply binary threshold at pixel value 230
identify connected components
compute perimeter and hull of each component

Figure 6.2: Pre-processing of topographic activation map images for quality measure computation. A: Blur and resizing effects on the original image when computing MSE AUC values. B: Connected component detection in the red and blue channel and obtaining perimeter and convex hull, specifically shown for detecting blue components in the red channel.

Component-Based Measures We further use two metrics that quantify topographic map quality based on the number and size of connected components in the images. The procedure can be followed in Figure 6.2B. First, we separate the image of the topographic map into the red and blue channel. For both channels, we apply a binary threshold at pixel value of 230 to separate blue or red regions (value 1) from the background (value 0), respectively. We empirically choose the threshold such that components agree with our subjective perception of regions in the topographic maps. Note that small values in the red channel indicate a blue region and vice versa. In the binarized channel images, we detect connected components using OpenCV [138]. We omit every component of size less than 10 px in a topographic map image of size 100×100 px by assigning it to the background component because they are not perceived as distinct regions. Finally, we compute two quality metrics. The first is the number of components, excluding the background, where fewer components indicate better distinguishable regions. In cases, where large parts of the topographic maps are white, the number of components alone can be an insufficient quality criterion. Therefore, we additionally compute the average size of these components as the second component-based metric. For more interpretable size values, we use the size relative to the largest possible circle area in a 100×100 px topographic map, which is $\pi \cdot 50^2 = 7854$ px.

Generally, the average component size decreases with the number of components. However, in the aforementioned case of largely white topographic maps with few components, a small component size indicates worse quality of the topographic map. Moreover, number and size do not account for whether the components are large but interwoven with others, which would make them more difficult to distinguish and compare. Therefore, we further compute the convexity of each connected component as the ratio of the length of its convex hull and its perimeter. In Figure 6.2B on the right, hull and perimeter are shown for multiple components. Using OpenCV, we compute the contour of each connected component with `findContours()` and the respective convex hull with `convexHull()`. We then compute the length of the contour (the perimeter) and the length of the hull with the OpenCV function `arcLength()`. We compute a size-weighted convexity swc for the red and blue channel, with $\mathbb{C}_{channel}$ as the set of connected components in the respective color channel as

$$swc_{channel} = \frac{1}{|\mathbb{C}_{channel}|} \cdot \sum_{C \in \mathbb{C}_{channel}} \left(\frac{size(C)}{\pi \cdot 50^2} \cdot \frac{hull(C)}{perimeter(C)} \right) \quad .$$

This means, we compute convexity as the fraction of the length of the convex hull and the perimeter of the connected component. Then, we aggregate convexity values of the components and reward larger convex components. To this end, we compute the fraction of total circle area (in a 100 × 100 px topographic map) occupied by each component and use these as weights for a weighted sum of convexity values. Lastly, we compute the average size-weighted convexity of the components in the respective channel.

Finally, we combine the two $swc_{channel}$ to a swc value depending on which channels contain connected components.

$$
\begin{aligned}
swc &= swc_{red} && \text{if} \quad \mathbb{C}_{blue} = \emptyset \\
swc &= swc_{blue} && \text{if} \quad \mathbb{C}_{red} = \emptyset \\
swc &= \frac{swc_{red} + swc_{blue}}{2} && \text{else}
\end{aligned}
$$

This results in a quality measure in the range of $[0, 1]$ and is optimal for two perfectly convex components in the red and blue color channel that jointly span the entire topographic map, including the possibility of a component in only one channel. We refer to this quality measure as “size-weighted convexity” in the following evaluation.

All Measures In total, we investigate five different measures: blur MSE AUC, resize MSE AUC, component count, average relative component area and size-weighted convexity.

Furthermore, we investigate the robustness of the quality of the topographic maps. To this end, we repeat each topographic map computation 100 times given the same input.

6.2.2 Topographic Map Quality for MLPs

For our simplest data set and model, MNIST and MLP, we compute NAPs in the first fully-connected layer, using the 10 classes as grouping. We then use the resulting NAPs to compute topographic maps with each of our 11 proposed layouting methods.

Pre-Selecting Layouting Methods First, we pre-select a subset of techniques that satisfy the qualitative expectations of the visualization described in Section 6.2.1. For the exemplary class “0”, we compare the methods with respect to the formation of regions of similar activations, the visual similarity to a topographic map in neuroscience and the ease of comparability.

Representativeness of Quality Measures Based on the pre-selected methods, we investigate which quality measure is best suitable to quantify the visual quality of topographic activation maps. To this end, we use a set of manually created topographic activation maps to evaluate whether the measures represent the perceived visual quality under different conditions. We then focus our evaluation on the most promising measure.

Quantitative Topographic Map Evaluation For the pre-selected methods, we investigate the quality of the resulting topographic maps in further detail. For each of the methods, we compute the quantitative evaluation measures described in Section 6.2.1. We include a random baseline layout to obtain an expected lower bound on the quality of the topographic maps. This random layout is a local force-only PSO which we initialize with random uniform values. For the most promising quality measure, we report the evaluation in the main part of the thesis. Results using all measures are provided in Appendix B.1.

6.2.3 Topographic Map Quality for CNNs

Topographic maps of CNNs represent feature maps, therefore, the color values are aggregated feature map NAPs. This difference to MLPs might affect which layouting method produces the best topographic maps. As for the MLP, we investigate visual quality of topographic maps for CNNs using the pre-selected layouting methods as well as a random layout. In addition, because of the need for aggregating feature map NAPs, we further investigate the four aggregation techniques “mean”, “max”, “center” and “random”. We perform our evaluation with CNNs trained on our MNIST variations shifted and padded MNIST, which are described in Section 4.1.1.

Layouting Methods As discussed in the NAP alignment evaluation (Section 5.3), pre-aligned data leads to the most descriptive NAPs. Therefore, as an initial quantitative evaluation, we compare the different layouting methods for the CNN trained on padded MNIST. This comparison involves 4 aggregation techniques, 4 quality metrics and 7 layouting methods across 100 repetitions of the layout computation. To reduce this large amount, we first create an overview comparison. In this overview, we only observe the mean qualities across the 100 repetitions. Further, we compute the average rank (1–7, best to worst) of the quality values of the layouting methods across the aggregation techniques. For example, if a method has the highest quality in all aggregation techniques, its average rank is 1. After this summary, we investigate the qualities in a detailed comparison. From this evaluation, we obtain the highest-quality layouting method which we use in the following evaluation steps.

MNIST Variations and Alignment Using the identified best layouting method, we compare the topographic map quality between the three different alignment variations of MNIST. With this comparison we investigate how the data alignment affects the topographic map qualities.

Aggregation Methods Again using the highest-quality layouting method, we compare the topographic map quality between the different aggregation techniques for each model, respectively. With this comparison, we investigate how strongly the choice of aggregation technique affects the topographic map quality. Also, we use this comparison to evaluate our expectation that the saliency-based alignment is particularly suitable for using a center value aggregation.

6.3 Results and Discussion

6.3.1 Pre-Selecting Layouting Methods

We first pre-select layouting methods that produce topographic maps which satisfy the qualitative criteria of Section 6.2.1. Topographic maps generated with each of the 11 methods for the MNIST MLP and the exemplary class “0” are shown in Figure 6.3. The figure shows the topographic maps as scatter plots to see the positions of the neurons and potential gaps in the layout as well as the interpolated visualization as the final topographic maps.

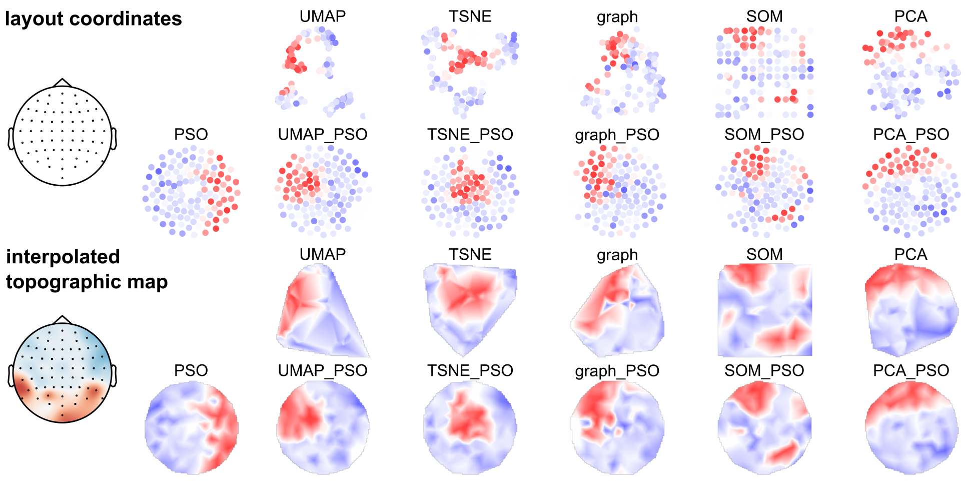


Figure 6.3: Topographic maps for one exemplary class for all proposed layouting methods. The scatter plots show the layouted neurons with NAP value-based coloring. Below are the resulting interpolated topographic maps. Left-most are examples of an electrode layout (top) and a topographic map (bottom) in neuroscience which we use as inspiration and qualitative target for our visualization. All layouts and colorings use the same class-based NAPs for an MNIST MLP model as input.

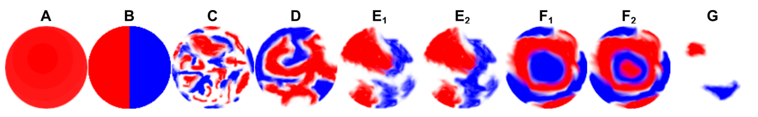
All methods can distribute the neurons to form regions of similar activations. Only the SOM technique splits up sets of co-activated neurons into multiple regions in the layout. This can happen because a SOM does not penalize the similarity of distant coordinates, for example, when initializing two distant positions with similar neurons.

Another criterion is that the neurons are well-distributed in the two-dimensional space. One reason for this criterion is that there are no empty regions in a topographic map of a brain either. In addition, a layout with varying neuron density leads to disproportionated regions in the interpolated topographic maps. This effect can, for example, be observed in the TSNE topographic map. The region of highly active neurons in the center is surrounded by areas without assigned neurons. In the interpolated image, the gaps cause the red region to be enlarged, which wrongly suggests that the highly active neurons are in the majority for this class. We observe the strongest neuron density variation for UMAP and TSNE, and the graph method leads to high density for groups of co-activated neurons. With the SOM and PCA methods, the neurons are well-distributed in the shown example but a higher variation of the neuron density can happen for different data, models or parameters, too. The best distribution is achieved with the PSO method, which is almost free of gaps in the layout and the density of neurons is similar across the whole layout. This observation is not surprising because the PSO method optimizes this property of the layout with the local force component.

For the images to be visually similar to topographic maps in neuroscience, we expect them to have a round shape. This property is particularly well satisfied with the PSO method, regardless of the initialization, which is again achieved by the local force. However, when initializing the PSO randomly, the quality of the topographic map is unreliable. In the shown example of a PSO topographic map, we observe several neurons of low activity within regions of high activity. This effect is likely related to the simultaneous optimization of activation similarity and neuron distribution that can interfere with each other. This supports our idea of first layouting the neurons by activation similarity, followed by distributing the neurons with a PSO using the local force only. Therefore, we conclude that the PSO methods with non-random initialization are the most promising techniques. We will use these methods for the quantitative evaluation and keep the randomly initialized PSO for comparison, as well.

6.3.2 Representativeness of Quality Measures

We manually created the topographic activation maps shown in Figure 6.4 as representative cases to see which quality measure most closely aligns with our expectations. Maps A and B represent ideal topographic activation maps, for which we expect the highest quality. With C, we represent a poor activation map that is difficult to interpret by visual inspection, so its quality should be correspondingly low. D is an example with few large components, however they are interwoven with each other and, therefore, we expect them to have less visual quality. E_1 and E_2 are more realistic examples of visually high-quality topographic maps, which differ in that the blue region is split by a small gap. Both should obtain good visual



		A	B	C	D	E_1	E_2	F_1	F_2	G
	blur MSE AUC	3.75	3.30	3.05	4.21	2.68	2.85	5.20	5.30	0.44
	resize MSE AUC	2.40	1.74	2.98	3.55	2.32	2.42	3.99	4.21	0.37
components	count	1	2	24	3	4	3	5	6	2
	average relative area	1.02	0.51	0.03	0.31	0.18	0.25	0.22	0.19	0.07
	size-weighted convexity	0.95	0.96	0.05	0.35	0.29	0.38	0.38	0.31	0.12

Figure 6.4: Comparison of different quality measurements on manually created topographic activation maps. Better scores for each quality measure are indicated by darker shades of green. Note that average relative component size and size-weighted are not 1 or 0.5 for the ideal examples A and B due to the pixel grid and approximations when obtaining connected components.

quality and the small gap should not strongly affect the value. F_1 and F_2 are examples to test whether the component-based scores can separate nested components correctly. For our last example G, which represents sparse regions in the topographic map, we expect a low quality that should be higher than that of C, as it contains more information.

Size-weighted convexity is most consistent with our expectations as it gives the highest quality to A and B, the lowest to C and G, and a higher quality to $E_{1,2}$ and $F_{1,2}$ than to the more convoluted activation map D. It does, however, not completely satisfy our expectations as there is little difference between the interwoven components in example D and the visually appealing topographic maps $E_{1,2}$. We suspect that this is due to a larger proportion of white areas in $E_{1,2}$ compared to D. Nonetheless, we consider the size-weighted convexity as the most suitable visual quality measure and use it for the following evaluation.

6.3.3 Quantitative Evaluation

In this section, we report the quantitative evaluation of visual quality of topographic activation maps using the size-weighted convexity measure.

MLP

We quantify the visual topographic map quality for the different PSO-based layouting techniques, The results for the first fully-connected layer of the MLP model trained on MNIST are shown in Figure 6.5.

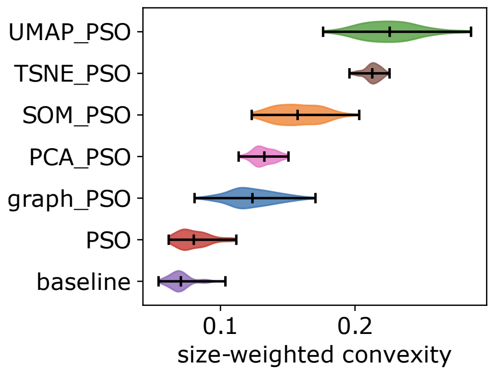


Figure 6.5: Quantification of the topographic map quality for MNIST MLP measured by size-weighted convexity. Each layout is computed 100 times and the plot shows the respective quality value distributions and their minimum, mean and maximum value.

We observe that all layouting techniques outperform the random baseline in the MLP. However, the default PSO is not significantly better than the baseline. SOM_PSO, PCA_PSO and graph_PSO are in the medium visual quality range. The best quality scores are obtained with TSNE_PSO and UMAP_PSO. UMAP_PSO performs better than TSNE_PSO on average and has higher variation between the runs, leading to both higher and lower qualities in individual layouts. Generally, the variation of the visual quality when computing the layout multiple times is surprisingly high for all methods. Especially for PCA_PSO which uses initial coordinates from the deterministic PCA, we expected more similar quality between the runs. We therefore suspect that a large part of the variation is attributed to the distribution of points after the initial layout. Likely, this is related to instabilities in the optimization if two particles get too close in an update and then repel each other too strongly due to the cubic influence of the distance in the repulsion force component. Because UMAP_PSO has the highest mean visual quality and its lowest-quality runs are better than all other techniques except TSNE_PSO, we find UMAP_PSO to be the most suitable layouting technique for MLPs.

The high visual quality of TSNE_PSO and UMAP_PSO also comes with a significantly longer computation time. Comparing the computation time to the PSO with random initialization, UMAP_PSO has around 27 times longer computation time and TSNE_PSO is 183 times slower than the regular PSO. For computing individual layouts, this time difference is negligible, but can accumulate. For our experiments with 100 repetitions, PSO needed 2 min, UMAP_PSO 43 min and TSNE_PSO almost 5 h. In practice, recomputing the same layout multiple times is unlikely. However, for example, computing layouts for several layers in a network multiplies the absolute time difference by the number of investigated layers.

CNN

First, we perform a quality comparison when using different layouting methods. In CNNs, the quality of the topographic maps can further be influenced by how the feature maps are aggregated to obtain color values, therefore, we also investigate different aggregation approaches. For some methods, random aggregation leads to extreme values such that differences between mean, max and center aggregation are not visible from the plots anymore. Therefore, we omit random aggregation for the comparison of layouting methods. Figure 6.6 shows the results of the quality comparison.

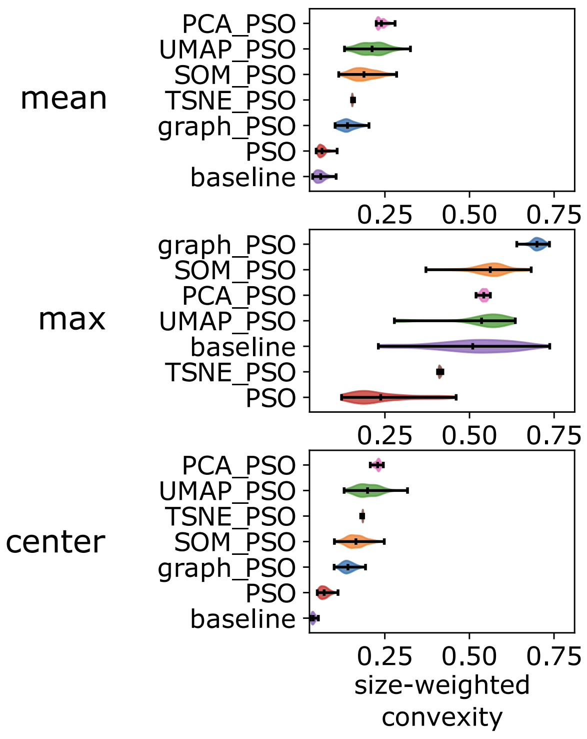


Figure 6.6: Topographic map qualities for padded MNIST for different layouting techniques using different aggregation functions. The plots show visual quality distributions over 100 repeated layout computations.

The maximum value of a feature map is typically a highly positive value, therefore, the topographic maps become large red regions. These lead to high visual quality values for every layouting technique. However, despite the high quality, the information content is low. Especially, the baseline layout can also lead to high quality in many cases, which indicates that maximum aggregation is not suitable. Mean and center aggregation, in contrast, yield more informative topographic maps, where the baseline is less qualitative than the layouts that are based on activation similarities.

As for MLPs, PSO with random initialization produces the lowest-quality topographic maps that are not significantly better than the baseline, and SOM_PSO and graph_PSO are in the medium quality range. However, in CNNs, PCA_PSO produces on average the highest quality values and TSNE_PSO only has medium quality for mean value aggregation. UMAP_PSO is second-highest average quality and it produces the individual layouts of highest quality.

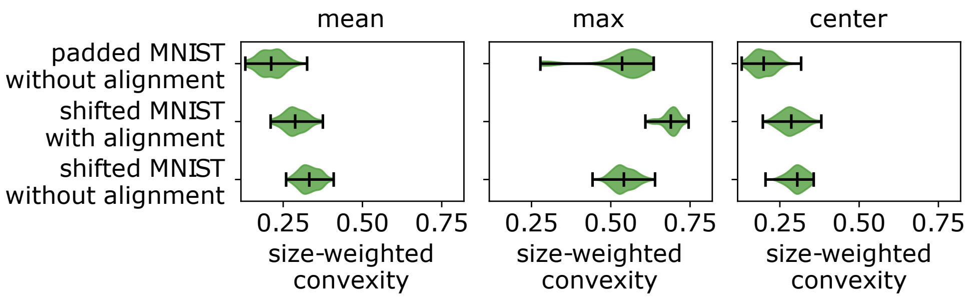
Despite PCA_PSO performing best on average for CNNs, we still consider UMAP_PSO as the better technique in general because it produces high-quality topographic maps for both MLPs and CNNs.

Alignment and Aggregation

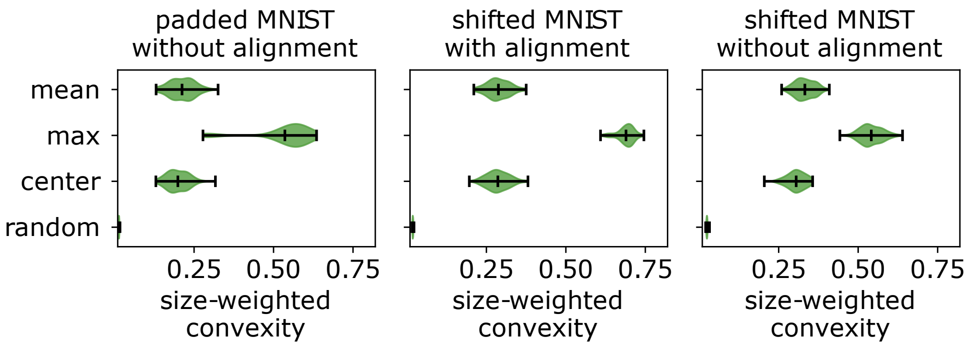
Creating topographic activation maps for CNNs differs from MLPs mainly in that feature map NAP values need to be aggregated to obtain colors. Here, we investigate whether there are differences in topographic map quality when using aligned or unaligned data and how suitable the different aggregation methods are in dependence of the alignment of the data. In the previous quality comparison of the methods, we found that maximum value aggregation is not informative for comparing different layouts. Nevertheless, we include it here in case that maximum aggregation performs better in certain alignment scenarios.

The topographic map qualities using UMAP_PSO as layouting technique are shown in Figure 6.7, from the perspective of alignment differences in Figure 6.7a and of aggregation method differences in Figure 6.7b.

Comparing the different MNIST alignment variations, we observe only small differences of qualities. Surprisingly, the pre-aligned data (padded MNIST) yields the lowest qualities and shifted MNIST without alignment



(a) Comparison of MNIST alignment variations.



(b) Comparison of aggregation methods.

Figure 6.7: Topographic map qualities using UMAP_PSO.

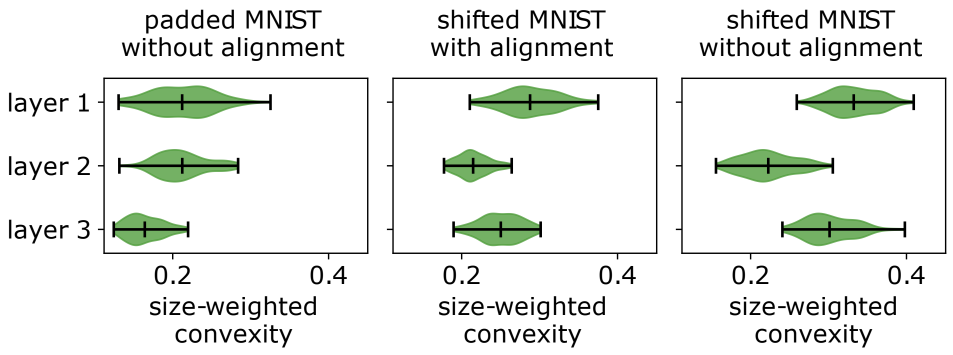
the highest. We suspect that the visual quality benefits from less informative values to show. Shifted MNIST without alignment produces more homogeneous NAP values due to stronger averaging out effects (compare Section 5.3). This in turn leads to regions that are less group-specific but satisfy the visual quality measure better.

All three tested aggregation functions outperform the baseline of picking a random feature map positions. As before, there is little difference between mean and center aggregation. Only for shifted MNIST without alignment, using mean value performs slightly better than center aggregation. Maximum aggregation again yields artificially high quality due to the large red regions, which are difficult to distinguish between the classes. Contrary to our expectation, there is no benefit of using center value aggregation when using our sensitivity-based alignment technique.

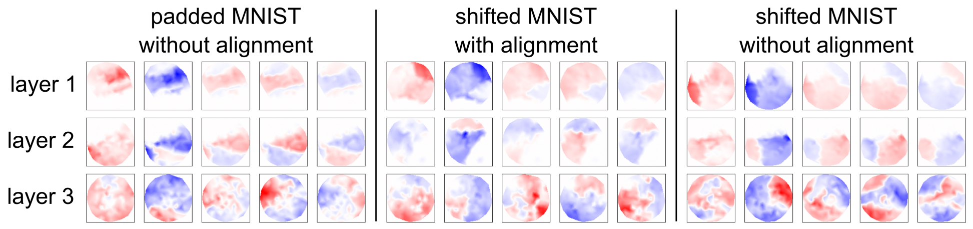
We conclude that maximum value aggregation is not suitable for producing color values for topographic activation maps. Both mean and center value appear similarly suitable, therefore, we decide to using mean aggregation in the following as it is the more common approach in literature. Further, we found that it is difficult to make a fair comparison because alignment variations and aggregation techniques influence the scale of the NAP values and how distinguishable the topographic maps are between the groups. Therefore, we expect similar issues in the following comparison of topographic maps in different layers due to value scale differences between layers.

Layer Comparison

Using the most promising layouting technique (UMAP_PSO) and mean aggregation, we further compare visual quality of topographic activation maps in different layers of the network. Results are shown in Figure 6.8a.



(a) Size-weighted convexity in different layers.



(b) Topographic maps for exemplary classes 0–4 in different layers. In the same layer and MNIST variation, topographic maps use the same color map.

Figure 6.8: Topographic maps and their visual qualities using UMAP_PSO and mean aggregation comparing the three convolutional layers in the CNNs for predicting padded and shifted MNIST classes.

For padded MNIST, quality values decrease with the depth of the layer. For shifted MNIST, both with and without alignment, the second layer decreases in quality but the third layer has higher quality than the second one. From visually inspecting the corresponding topographic maps (see Figure 6.8b), the qualitative difference between padded and shifted MNIST is difficult to detect. In layer 1, padded MNIST has larger white regions than shifted MNIST, that means, more unspecific neurons, which leads to lower visual quality according to our measure. In layer 3, the differences are not as obvious but it appears as if regions in shifted MNIST are of slightly higher contrast between under- and over-active neurons than for padded MNIST. We suspect that the higher visual quality is related to that the shifted MNIST data influence on the learned patterns by the model. The differently located MNIST digits in shifted MNIST provide richer training information than padded MNIST where all digits are exactly in the center such that the model learns more representative features.

6.3.4 Comparison to Other Visual Quality Measures

We decided to use size-weighted convexity based on only a few manually created examples. In the case that these examples are not representative enough, we also perform the quantitative evaluation for all other investigated visual quality measures, as well. Here, we only summarize the results and provide the detailed results and discussion in Appendix B.1.

Comparing the layouting techniques, the component measures (count and average relative size) agree with size-weighted convexity that UMAP_PSO is the highest-quality technique. Both MSE AUC quality measures consider UMAP_PSO and TSNE_PSO as similarly high quality. Consequently, UMAP_PSO yields topographic activation maps of high visual quality according to all investigated measures, both for MLPs and CNNs.

Regarding the influence of alignment and aggregation techniques, we observe similar qualities according to the number and size of connected components. For the blur and resize MSE, however, the quality is generally low for shifted MNIST with alignment. Another clear difference for blur and resize MSE quality is that the aggregation function does not substantially affect the quality for shifted MNIST without alignment. There is, however, a change in visual quality, which supports that the blur and resize MSE qualities do not represent visual quality well. Similarly, the findings for the layer comparison are supported by the connected component count and size but not by the blur and resize MSE.

In conclusion, we find that the image perturbation quality measures do not reflect subjective visual quality well and disagree with the connected component-based measures in several cases. The combination of component size and number agrees with the findings of the size-weighted convexity. Therefore, we still consider the size-weighted convexity as the most suitable quality measure.

6.4 Chapter Summary

In this chapter, we presented topographic activation maps to visualize high-dimensional NAPs as a two-dimensional representation.

In essence, our visualization approach is a dimensionality reduction technique. However, in addition to reducing the dimensionality of the data while retaining as much information from the high-dimensional data space as possible, we aim to achieve a visual appearance that resembles how neuroscience visualizes the activity of the brain. We found that neither common dimensionality reduction techniques (like tSNE or UMAP) nor designated topography-learning approaches (like SOM or our PSO technique) satisfy both requirements on the neuroscience-inspired visualization of activations. However, first using dimensionality reduction to project high-dimensional NAPs into a two-dimensional space and then using PSO to achieve a neuroscience-typical round shape yielded the best results.

Among these two-step methods, we found the UMAP_PSO layouting technique to be best suitable for producing topographic maps. This method first applies a UMAP dimensionality reduction and then uses a local-force only PSO to obtain a round shape of the visualization. We evaluated the topographic map quality over different model types, ways of aligning data, and aggregation techniques for reducing feature map NAPs to a single value for the visualization.

Importantly, the visual quality of topographic maps does not imply that they represent useful information about the DNN activations. The representativeness needs to be provided by meaningful NAPs, which is discussed in Chapter 5.

Visualizing NAPs as topographic activation maps works best for layers that have many neurons or feature maps. Otherwise, the visualization has a very low resolution, for example, in smaller networks or with toy examples. Out of this reason, our MNIST classifiers for evaluating topographic maps use 128 feature maps although fewer would have been sufficient for such simple data set. Secondly, topographic activation maps are most useful for visually comparing different groups. However, if the number of groups is large, it can become difficult to compare the large number of resulting visualizations. To counteract this problem, when visualizing topographic activation maps of multiple classes, we recommend to order them by the class activation similarity. We provide details and examples of this approach in Chapter 7, particularly Sections 7.3 and 7.4.

Finally, topographic maps are difficult to fairly compare between different models or different layers within the same model. Both visualization and the corresponding quality values depend on the number of (active) neurons and the scale of the NAP values, such that a comparison is usually not conclusive. Therefore, we recommend to only compare groups in the same model and layer and only contrast these relations the groups' relations when observing different layers and models. For example, in Section 7.4, we compare topographic maps between groups in different layers and discuss how the similarities and differences of the groups change compared between an early and a deep layer.

7 Application

This chapter addresses how NAPs and topographic activation maps can be applied to get insight into trained DNNs. First, we discuss how well our technique scales to different architectures and data sets (Section 7.1). Then, we show exemplary applications on using NAPs (Section 7.2) and topographic activation maps (Sections 7.3 and 7.4).

7.1 Scalability

7.1.1 Architecture – Deep Models

Computing NAPs and visualizing them as topographic activation maps is performed layer-wise. Therefore, it is straight-forward to apply it to models of any depth. However, analyzing all layers becomes computationally highly expensive for deeper models and it is necessary to only focus on a subset of layers of interest. Our recommendations about how to select these layers is given in Section 5.1.1. An example of computing NAPs for DNNs with many layers is given in Section 7.4, where we analyze a VGG16 model. Further, in Chatterjee et al. [18], we perform a NAP analysis of a set of five large models, ResNet18, ResNet34, InceptionV3, InceptionResNetV2 and DenseNet161, applied to classify respiratory diseases in chest X-ray images.

7.1.2 Architecture – Wide Layers

The width of a layer describes its number of neurons or feature maps. The consequently larger dimensionality of the activations affects the computation of NAPs because more values need to be processed. However, this increase in computational effort scales linearly with the width and can be parallelized, therefore, does not impact the applicability of our method. The memory usage for wide layers is limited by the GPU memory to run the model in the first place, not the the memory required to run the NAP analysis. As the width only influences the dimensions of the NAPs, representational similarity analyses can also be performed for wide layers.

Computing topographic maps of NAPs, on the other hand, is strongly affected by the layer width because it requires layouting the different neurons or feature maps in a layer. In particular, there are several operations that perform pairwise comparisons (by computing similarity matrices) between the neurons or feature maps. This leads to quadratically growing number of comparisons with respect to the width of a layer. Especially the non-linear dimensionality reduction techniques UMAP_PSO and TSNE_PSO are affected because of their generally longer runtime.

7.1.3 Architecture – Large Feature Maps

Large feature maps typically result from high-resolution input data but can also occur in models that use transposed convolutional layers to upsample representations. The computation of NAPs linearly scales with the size of the feature maps because it adds more values to compute an average of. As with wide layers, the memory usage increases, but is already limited by GPU memory during the model training and inference.

The main difficulty of large feature maps is the aggregation of feature map values when creating topographic maps. For this visualization, feature maps are reduced to a single value. This can heavily affect how representative this value is of the feature map NAP content. Commonly, this issue is addressed by not considering the entire input at once but to observe patches of it, for example, in [17]. Investigating this option for NAPs is outside the scope of this thesis but is an interesting future research direction.

7.1.4 Data – Large Groups

Using groups with many examples is possible with our techniques but comes with several disadvantages. Firstly, the computation time for obtaining the activations and averaging them increases with larger number of examples and potentially needs a memory-efficient implementation to handle all instances. Secondly, a large set of instances is unlikely to provide an useful representation of the group. Either there is much redundancy in the group, leading to wasteful computation, or the variation between the instances in the group is so high that the averaging removes too much information. Therefore, using large groups is generally less suitable for our approach.

7.1.5 Data – Large Number of Groups

Considering the same data set with groupings of different number of groups, there is almost no difference in how much computation is necessary. In each case, activations are computed for each instance and are involved in exactly one averaging operation. Only when normalizing the NAPs with an average over the NAPs of the groups, computing the normalizer is negligibly more demanding. Hence, the NAP computation effort does not depend on the number of groups.

However, comparisons of groups, for example, using a representational similarity analysis with clustering, become more computationally expensive for large numbers of groups because they consider their pairwise relations. In addition, the visualization of the NAP analysis results of many groups can become very cluttered. Later in this chapter (Section 7.2.2), we show an example of 62 groups (phonemes), which is already difficult to visualize without providing close-up views of subsets. Further, comparing topographic activation maps of many groups is complicated as they are two-dimensional images that need much space in the visualization.

To work with many groups while still producing visualizations that are not too cluttered, we recommend to facilitate group similarities. Firstly, ordering the groups by their similarity with hierarchical clustering helps to put similar groups in the vicinity of each other. This way, the order of the groups itself helps to visually identify group similarties. In addition, similarities of NAPs or topographic maps are easier to detect because the corresponding groups are close to each other. For very large numbers of groups, it is further useful to subdivide visualizations, for example, by splitting it according to branches from the hierarchical clustering result to obtain subsets of similar groups. Some data sets might provide hierarchical labels that can be used for subdividing instead of a clustering. For example, the ImageNet data set [157] uses the WordNet hierarchy to provide labels, and TIMIT [41] provides information about which high-level category each phoneme belongs to.

7.2 NAP Analysis of Automatic Speech Recognition Models

In this experiment, we show how to use NAPs to analyze phoneme and letter representations in fully-convolutional ASR models. The experiment has been performed as part of one of our publications [89] where we refer to this version of NAPs as SNAPs. However, for consistency in the thesis, we use the term NAPs, while noting that it uses saliency information to align the input and activations and to mask out prediction-irrelevant information.

We perform this experiment for TIMIT because we require the phoneme annotations for our evaluation. In our previous work Krug et al. [88], which is not part of this thesis, we also show that the analysis is possible for the large-scale speech data set LibriSpeech [142].

7.2.1 Experiment

Plotting NAP

For data that are visually interpretable, input layer NAPs are interpretable, as well. Although understanding spectrograms requires some domain knowledge, they can be visually interpreted to some extent. However, due to the normalization, input layer NAPs are not spectrograms but describe how the intensities of the frequencies of the represented group differ from the rest of the data. Plots of hidden layer NAPs are interpretable in fewer cases because it requires that the hidden layer activations can be interpreted themselves. To a certain extent, this is possible for 2D CNNs applied on visually interpretable data. In such case, NAPs that correspond to single feature maps can be plotted and inspected. In the 1D CNN that we use in this work, feature maps are only one-dimensional and, in contrast to the frequency dimension in the input, do not have a meaningful order. Hence, in this experiment, we demonstrate visual inspection of NAPs only for the input layer.

We use the W2L model to evaluate visualizations of input layer NAPs. To this end, we first compare input layer NAPs of letters with simple normalized aligned averages, to show the advantage of using saliency map information. Secondly, we evaluate whether NAPs of phonemes reveal representative patterns by qualitatively comparing exemplary input layer NAPs with expected phoneme-typical patterns.

Representation Power of Layers for Different Groups

In our earlier work [88], we applied hierarchical clustering with Euclidean distance and complete linkage [126] to NAPs of letters and phonemes, using a fixed threshold for the emergence of clusters. Here, we investigate the clustering approach using different distance thresholds at percentiles 87 % to 96 % in steps of 3 %. We empirically found these thresholds to lead to different numbers of clusters for our model. Applying the clustering algorithm to other models might need other thresholds if the range or distribution of distance values are different due to model properties, for example, the used activation function. For quantifying the quality of the clustering result, we compute its Silhouette score [155]. The Silhouette score is a measure of how similar instances in the same cluster are to each other compared to their similarity to instances of the nearest other cluster. In the context of NAPs, a high clustering quality in terms of Silhouette score indicates that there are sets of groups, where groups in the same set activate neurons similarly and groups of different sets activate neurons differently. The resulting sets of groups are potential high-level concepts that the network encodes in the respective layer. However, high clustering quality does not guarantee that this concept is meaningful to a human. Therefore, evaluating whether the clusters are interpretable sets of groups still requires to manually inspect the clusters. In addition to investigating the Silhouette scores at individual distance thresholds, we further investigate the average Silhouette score over the four used thresholds in each layer.

We evaluate this approach by contrasting the W2L model with the models that we train with transfer learning (compare Section 4.2.4). Through transfer learning, the five bottom layers of the models W2L_TL_frozen and W2L_TL_finetuned are pre-trained to predict phonemes. Therefore, we expect the best clustering of phonemes to emerge at this depth of the networks and to be better than in the W2L model. This expectation does not involve that the clusters correspond to interpretable concepts like phonemic categories. Further, for the W2L model, we compare the clustering of NAPs between grouping by predicted letter and annotated phoneme.

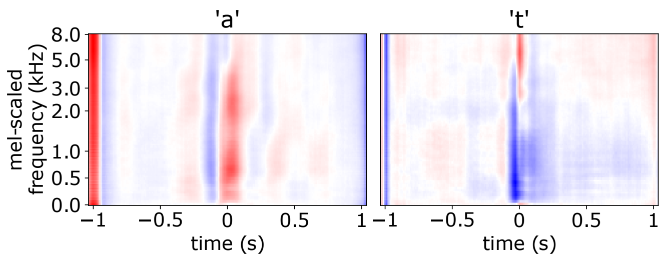
7.2.2 Results

Per-layer NAPs

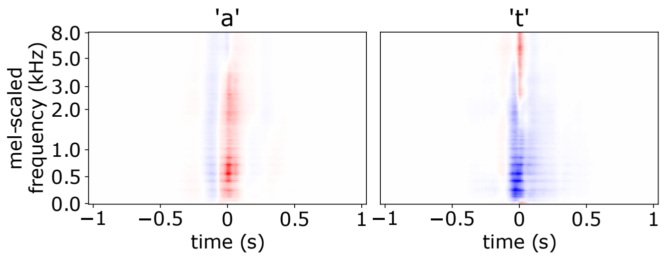
NAPs in the input layer are directly interpretable. They show how the intensity of frequencies differs from the average over the complete data set, indicated with red and blue color for higher and lower intensity, respectively. The sensitivity-based masking causes that the NAPs only show regions which are important for the prediction. This advantage is demonstrated comparing the top and middle row in Figure 7.1. While without the mask, both shown NAPs show a pattern over the whole receptive field size, the corresponding masked NAPs also identify prediction-relevant parts of the pattern. In addition to providing more information about what part of the input the model uses for prediction, padding artifacts that occur as a result from the alignment procedure are masked as well. An example for these padding artifacts are the high values at -1 s in the pattern of letter 'a' in Figure 7.1a.

We observe phoneme-typical patterns in the input layer NAPs (Figure 7.1c). Phonemes `aa` and `ae` share a high intensity formant at around 700 Hz. A second formant is identified at around 1200 Hz and 1900 Hz for `aa` and `ae`, respectively. The input pattern for phoneme `t` shows a change of high to low intensities of all frequencies at the alignment time. The patterns for all three observed phonemes match the expectation well. A main difference to the expected patterns is that identified formants spread a wider range of frequencies, which is likely an effect of averaging recordings from different speakers.

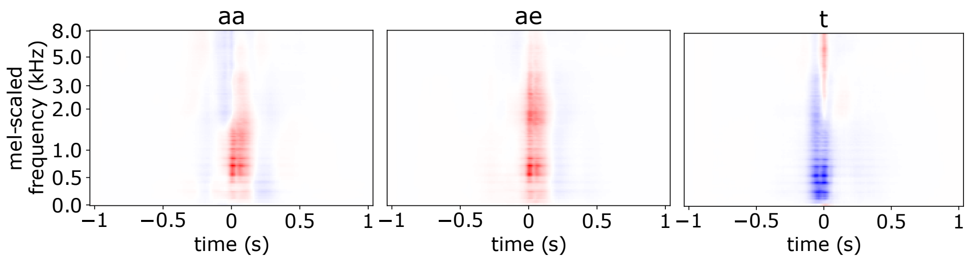
The input layer NAP for letter 'a' is more similar to the input layer NAP of phoneme `ae` than to the NAP for phoneme `aa`. This indicates that letter 'a' is pronounced as `ae` more frequently than as `aa` and that the NAP of letter 'a' is dominated by the over-represented pronunciation `ae`. This dominating effect can happen for groups with high variation between the instances. Therefore, it particularly affects unbalanced groups in which a subset of instances is more homogeneous than the other instances. Because of this,



(a) Input layer NAPs of letters ‘a’ and ‘t’ without saliency mask.



(b) Input layer NAPs of letters ‘a’ and ‘t’ with saliency mask.



(c) Input layer NAPs of phonemes `aa`, `ae` and `tt` with saliency mask.

Figure 7.1: Comparison of input layer NAPs with and without masking prediction-irrelevant positions in model W2L for letters and phonemes. The plots visualize the difference of intensity of frequencies between group-average and the average over the complete data set. White indicates zero difference, red and blue color indicate higher and lower intensity, respectively. Lighter color also indicates small importance for the prediction. Colors represent the same values only for subfigures in the same row.

NAPs are best suitable for groups with small intra-group variance. Consequently, in this work, we compute NAPs for individual phonemes, instead of using higher-level groups like vowels or fricatives. Input layer NAPs in the model W2L are shown in the appendix for all annotated phonemes in Figures A.10 and A.11 and for predicted letters in Figure A.9.

In deeper layers, the order of feature maps is uninformative because there are no connections between feature maps in the same hidden layer. Therefore, the corresponding NAPs cannot be interpreted by visual inspection.

In all layers, we observe that NAP values become smaller and drop to 0 the further away they are from the alignment time. This indicates that the model does not use the complete receptive field for the prediction of the majority of instances. Thus, it is possible to compress the model, for example, by choosing fewer layers or filters or by decreasing the kernel sizes.

Evaluation of the Representational Similarity

We evaluate whether NAP clustering indicates representational quality. To this end, we first investigate Silhouette score correspondence to a ground truth obtained through transfer learning and, based on the findings, inspect the emerged clusters in specific layers. For this evaluation, we compare the three models W2L, W2L_TL_frozen and W2L_TL_finetuned (compare Section 4.2.4).

Silhouette Scores Figure 7.2 shows an overview of the Silhouette scores for clusterings in all layers of the used models at different distance thresholds as well as averaged over all the used distance thresholds. In general, higher Silhouette scores indicate better clustering quality. By increasing the distance threshold, the number of clusters decreases.

Regarding the average Silhouette scores, the models do not differ substantially. We observe the highest difference between any two models in the output layer, where the W2L model has an average Silhouette score of 0.31 and the W2L_TL_frozen model only 0.21. The second largest difference is in layer 5, where the Silhouette score is higher for the TL models (frozen: 0.15, finetuned: 0.16) than for the W2L model (0.09).

The non-averaged Silhouette score curves are more clearly distinguishable between the transfer learning models and the W2L model. Using smaller distance thresholds (87th and 90th percentile), the Silhouette scores are higher for the W2L model than for the transfer learning models, but the difference between the clustering qualities of the models varies between layers. However, for the W2L model, we do not observe a change in clus-

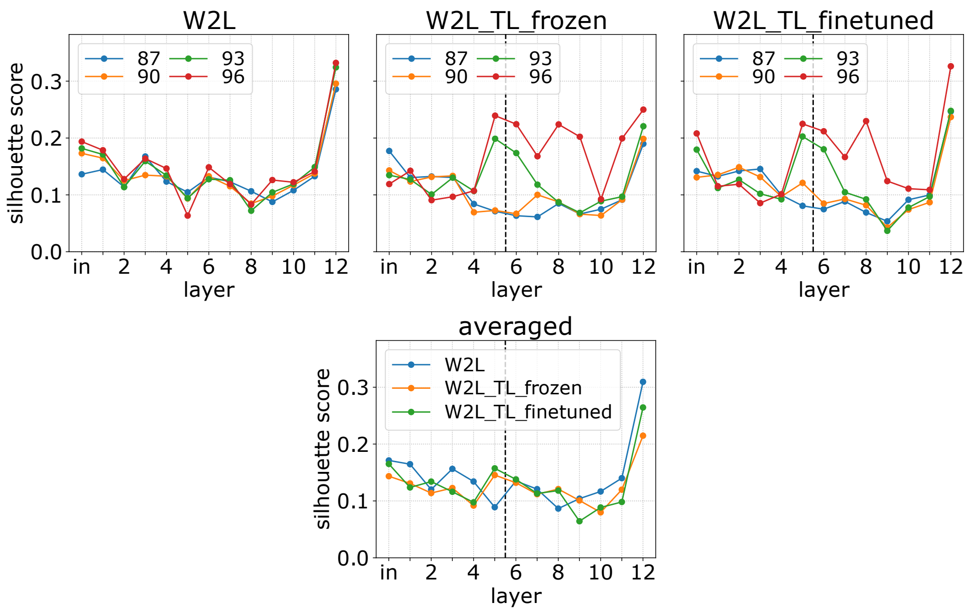


Figure 7.2: Silhouette scores at different distance thresholds. Through transfer learning, models W2L_TL_frozen and W2L_TL_finetuned are expected to have best phoneme encoding in layer 5. Transition from pre-trained layers to added layers is indicated by a vertical dashed line.

tering quality when increasing the distance threshold. This is different to the transfer learning models, in which Silhouette scores increase strongly from layer 4 to 5 when using higher distance thresholds (93$^{\text{th}}$ and 96$^{\text{th}}$ percentile). At the same layer, the W2L model decreases in Silhouette score. We consider this decrease a random co-occurrence because it is independent of our choice of the depth of the transfer learning base model.

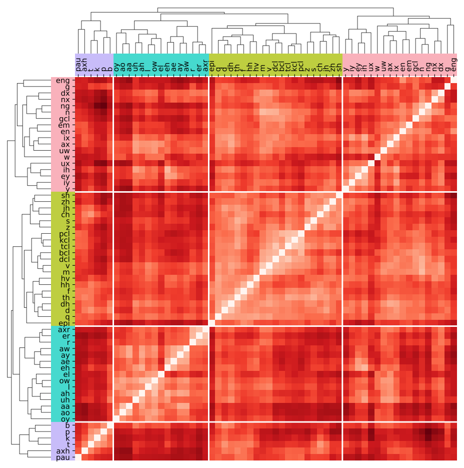
We expect that phonemes are best represented in the fifth layer of the transfer learning models because it is the deepest layer which is pre-trained on phoneme prediction. For the higher distance thresholds, the Silhouette scores correspond to this encoding that is induced through pre-training. This demonstrates that clustering quality can be used to investigate representation of groups. However, the result is sensitive to the choice of parameters for the clustering algorithm. We therefore recommend to not rely on a single parameter setting but to perform the clustering evaluation with different parameters although it multiplies the computation time.

Emerged Clusters In addition to only observing the Silhouette score, we investigate the emerged clusters in more detail. As described before, there is a high change in Silhouette score in both the W2L_TL models and the W2L model from layer 4 to 5, which is the highest for the clustering at the 96th percentile. Therefore, we investigate the clustering result at the 96th percentile threshold in layers 4 and 5 in more detail. As the frozen and finetuned model do not differ substantially, we only compare W2L and W2L_TL_frozen. A visualization of the NAP clustermaps is shown in Figure 7.3 and an enlarged view of the corresponding clustering assignments is shown in Figure 7.4. In the following, we will refer to clusters of at least three groups “main clusters” and focus on them because they indicate groups of similar representation.

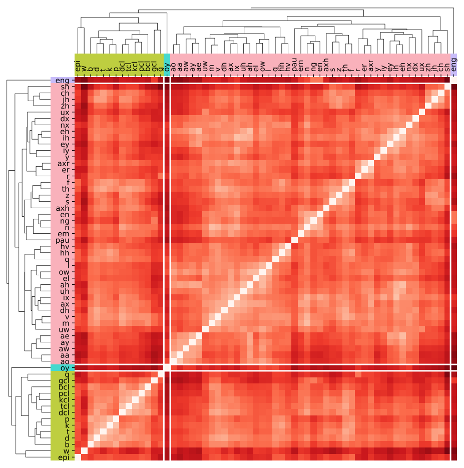
For W2L_TL_frozen, two distinguishable main clusters emerge in layer 4, while in layer 5, there is only a single large cluster. Observing a single large cluster with high pairwise similarity values indicates good representation on the level of individual phonemes. Because the model distributes phonemes evenly in the representation space, it is able to distinguish between each of the individual phonemes. This is our expected result because the fifth layer of the W2L_TL_frozen model is the deepest layer that is pre-trained on phoneme prediction trough the transfer learning approach. Decreasing the clustering distance threshold in the transfer learning model forces the clustering to divide the homogeneous cluster, leading to significantly smaller Silhouette scores.

The W2L model, in contrast, shows more distinct subclusters in the 4th and 5th layers. The number of clusters is the same in both layers but the sets of phonemes being assigned to the same cluster is substantially different. 35.4°% of phoneme pairs change from being assigned to the same cluster to being in different clusters or vice versa. At the same time, distances between NAPs in the W2L model are much higher and vary stronger than in the transfer learning model. Phoneme clusters in the W2L model indicate that the model encodes high-level concepts of similar sounds. For example, the red cluster in the 5th layer (Figures 7.3c and 7.4c) contains most vowels.

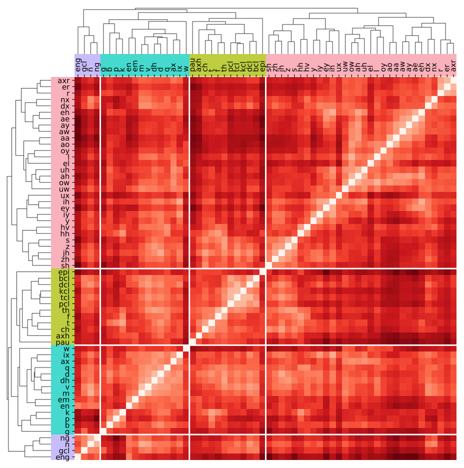
Comparing Letter and Phoneme Representations in W2L To demonstrate another application of NAP clustering, we investigate whether the W2L model implicitly encodes phonemes as intermediate representation for predicting



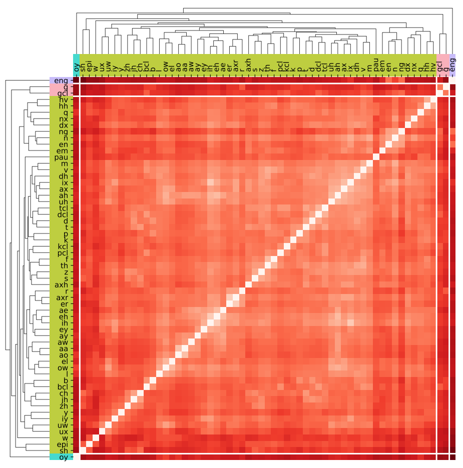
(a) W2L, layer 4, 96th percentile



(b) W2L_TL_frozen, layer 4, 96th percentile

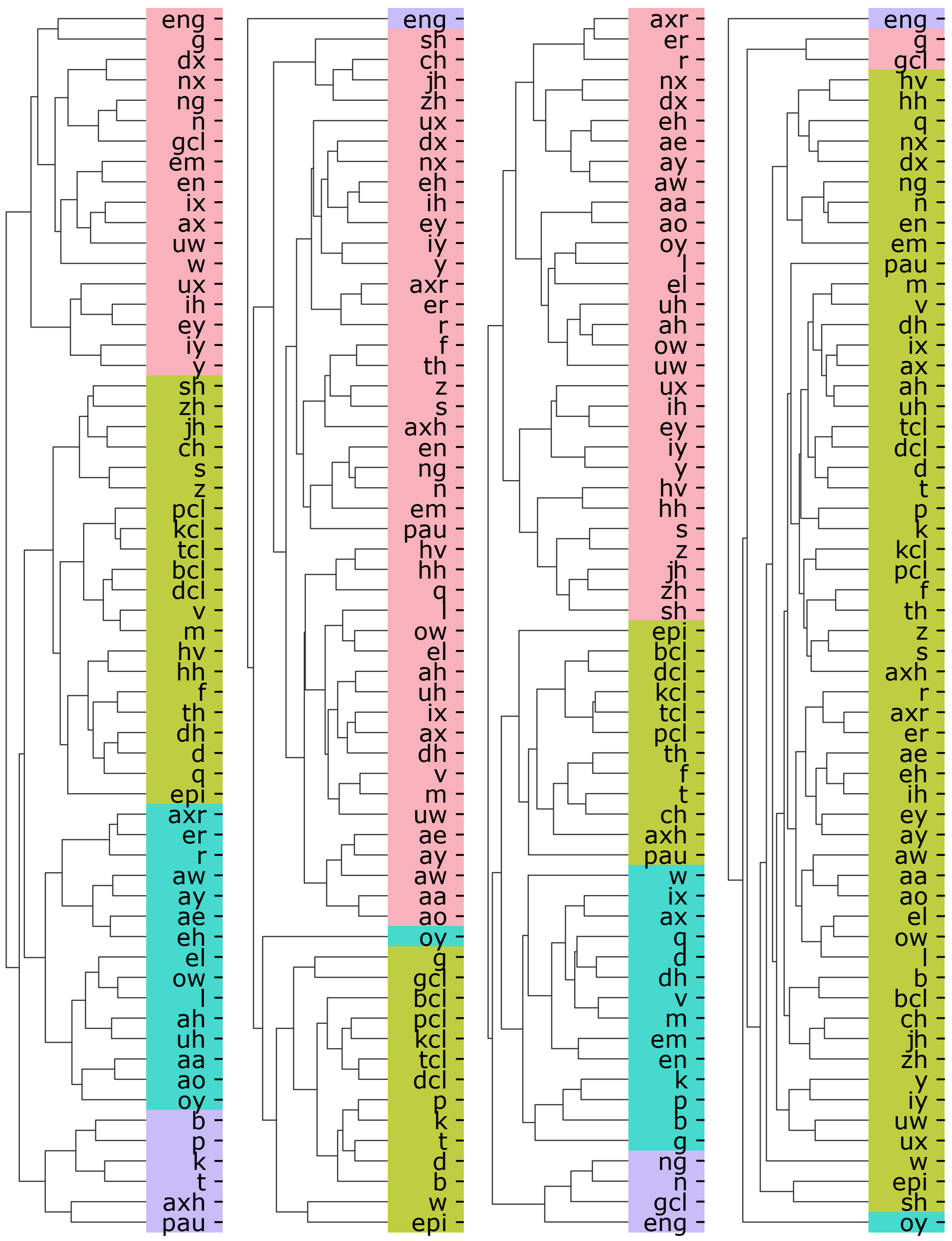


(c) W2L, layer 5, 96th percentile



(d) W2L_TL_frozen, layer 5, 96th percentile

Figure 7.3: Clustermaps for W2L (left) and W2L_TL_frozen (right) in layer 4 and 5 (top and bottom) at the percentile with highest change of Silhouette score from layer 4 to layer 5. Darker shades of red indicate higher distance. Heat map colors do not represent same distance values in different plots. Colors of clusters in different subfigures do not represent mapping of the clusters. See Figure 7.4 for an enlarged clustering view.



(a) W2L layer 4 96th percentile

(b) W2L_TL_frozen layer 4 96th percentile

(c) W2L layer 5 96th percentile

(d) W2L_TL_frozen layer 5 96th percentile

Figure 7.4: Clustering result for W2L (a, c) and W2L_TL_frozen (b, d) compared between layer 4 (a, b) and 5 (c, d) at the percentile with highest change of Silhouette score from layer 4 to layer 5. Clustering assignment is shown by coloring. Colors of clusters in different subfigures do not represent mapping of the corresponding clusters. Pairwise distances are visualized in Figure 7.3.

letters. First, we compute NAPs for the W2L model using phonemes and letters as grouping, respectively. Grouping by letter uses the predictions of the model and phoneme grouping is based on the annotated phonemes at the center time step of the input spectrogram frame. We then cluster both sets of NAPs in each layer and compute the Silhouette scores at different distance thresholds. In the input layer, we cluster the spectrogram frames. Figure 7.5 shows the result for letters, phonemes and the averages over distance thresholds for both groupings.

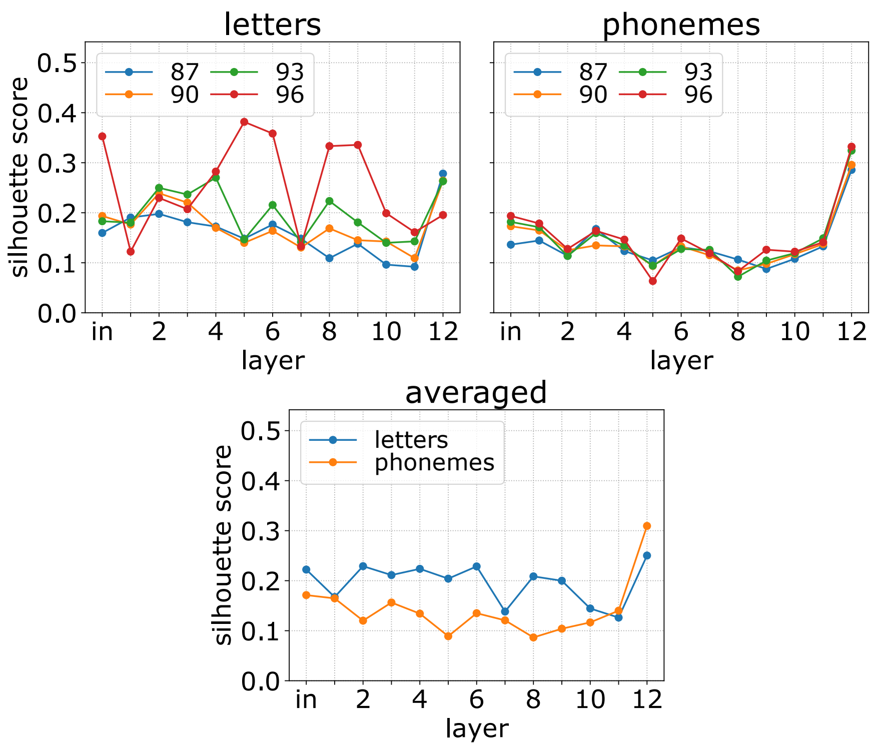


Figure 7.5: Silhouette scores at different distance thresholds, derived from NAP clustering in the W2L model comparing grouping by predicted letters and by phoneme annotation. ‘in’ denotes the input layer.

For letters, the clustering quality strongly depends on the chosen distance threshold, while the threshold has only minor influence on the phoneme clustering quality. Particularly, clustering at the 96th percentile of letter NAPs differs from clustering at smaller distance thresholds. Compared to the evaluation experiment in the previous Section 7.2.2, the observed Silhouette score curves are not as conclusive. According to the Silhouette scores, letters are clustered better than phonemes in all except the two final layers. This is surprising as we expect a letter prediction model to specifically cluster letters well in the output layer. Moreover, letter clustering has the highest Silhouette score in layer 5 (at 96th percentile), while phoneme clustering has its minimum clustering quality in this layer.

Because layers 5 and 12 stand out in terms of clustering quality, we investigate clustering in these layers in more detail. We particularly compare the NAPs for letters and phonemes in the W2L model, using the respective distance threshold with the highest respective clustering quality. A detailed visualization of the pairwise distances between the NAPs as clustermaps is shown in Figure 7.7, while Figure 7.8 shows the corresponding clustering assignments as enlarged view.

Although the Silhouette scores indicate better clustering for letters in layer 5, only a single-letter cluster for ‘j’ is distinguishable from all other letters. In addition, several similarities do not reflect interpretable concepts. For example, unexpectedly, letter ‘r’ is most similar to letters ‘a’, ‘o’ and ‘u’. We similarly observe a single large cluster for the transfer learning model in the previous evaluation experiment (Section 7.2.2, Figure 7.3d). The main difference is that, here, the distance values between letter NAPs are much higher than the distances between phoneme NAPs. In this case, we suspect that the high clustering quality is an artifact from distinguishing the NAP of ‘j’ as the rarest group (only 160 frames are predicted as ‘j’). Emerging phoneme clusters in layer 5 represent more meaningful groups, although having lower Silhouette score. The pink cluster encompasses nasals (`ng`, `n`, `eng`). Green contains multiple fricatives (`s`, `z`, `zh`, `sh`). Purple only consists of plosives (`k`, `p`, `b`, `g`). Red, orange and yellow are mainly grouping different similar sets of vowels (and semivowels). See Figure 7.6 for a graphical overview of which phoneme categories are associated with which cluster.

In the output (12th) layer, we observe more meaningful clustering of letters, for example, a cluster containing all vowel letters and a cluster of sibilant-typical letters (blue and yellow clusters in Figure 7.7c, respectively). However, other letters that we expected to be close (for example, ‘p’ and ‘b’ or ‘t’ and ‘d’) are far apart from each other. We hypothesize that this indicates how certain the model is when predicting particular letters. For example, ‘p’ and ‘b’ might be easy for the model to distinguish, which leads to output layer activations and corresponding NAPs that are more distant from each other. In the previous Section 7.2.2, we observe a similar pattern

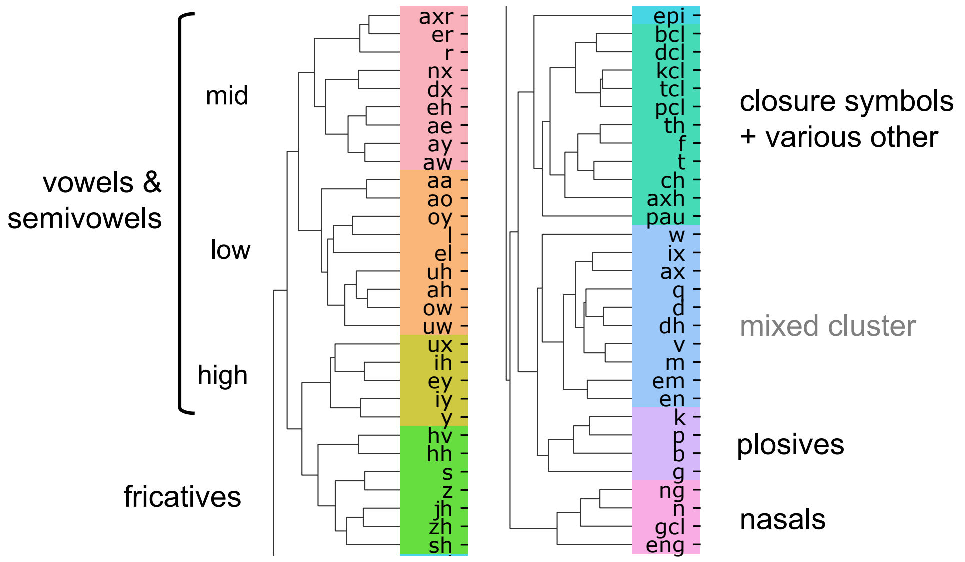
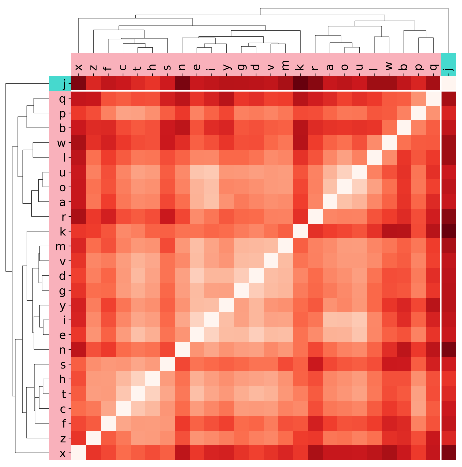


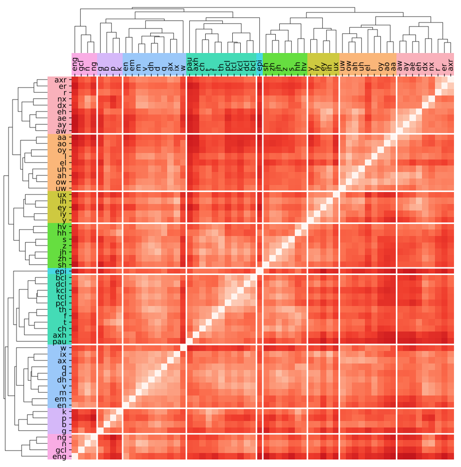
Figure 7.6: Phoneme clusters in layer 5 of the W2L model. We manually inspected the clusters and identified overrepresented phoneme categories in the respective clusters. That means, not all clusters exclusively contain phonemes of the shown categories.

for the phoneme clustering in the fifth layer of the W2L_TL_frozen model (Figure 7.3d). There, we suspected that the W2L_TL_frozen model can easily distinguish between the phonemes in the fifth layer and therefore distributes them more evenly in the representation space. For letter prediction in the W2L model, however, it appears to be harder to distinguish between some of the letters, which leads to the corresponding output layer NAPs being more similar to each other. We conclude that this difference of the clustering results reflects that it is more difficult to distinguish letters than phonemes, which is reasonable considering the high variability of how letters are pronounced.

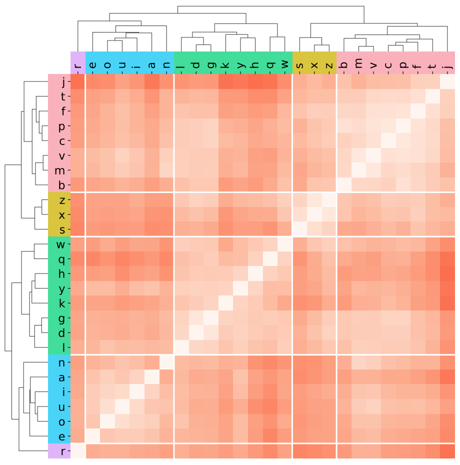
For phonemes in layer 12, emerging clusters are more distinct from each other, both compared to the letter clustering as well as compared to phoneme clustering in layer 5. Phoneme pau as pause marker is clearly distinguishable from the other phonemes. Apart from that, we observe that the clustering of phoneme NAPs in layer 12 is worse than in the fifth layer. The emerged clusters do not represent any phonemic category both in terms of their size and the phoneme similarities. This observation supports our expectation that phonemes are worse represented in deeper layers of



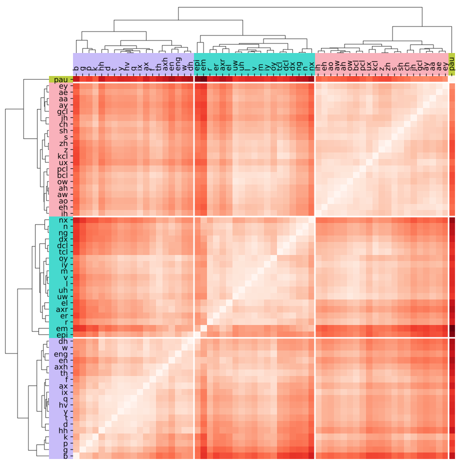
(a) letters, layer 5, 96th percentile



(b) phonemes, layer 5, 87th percentile

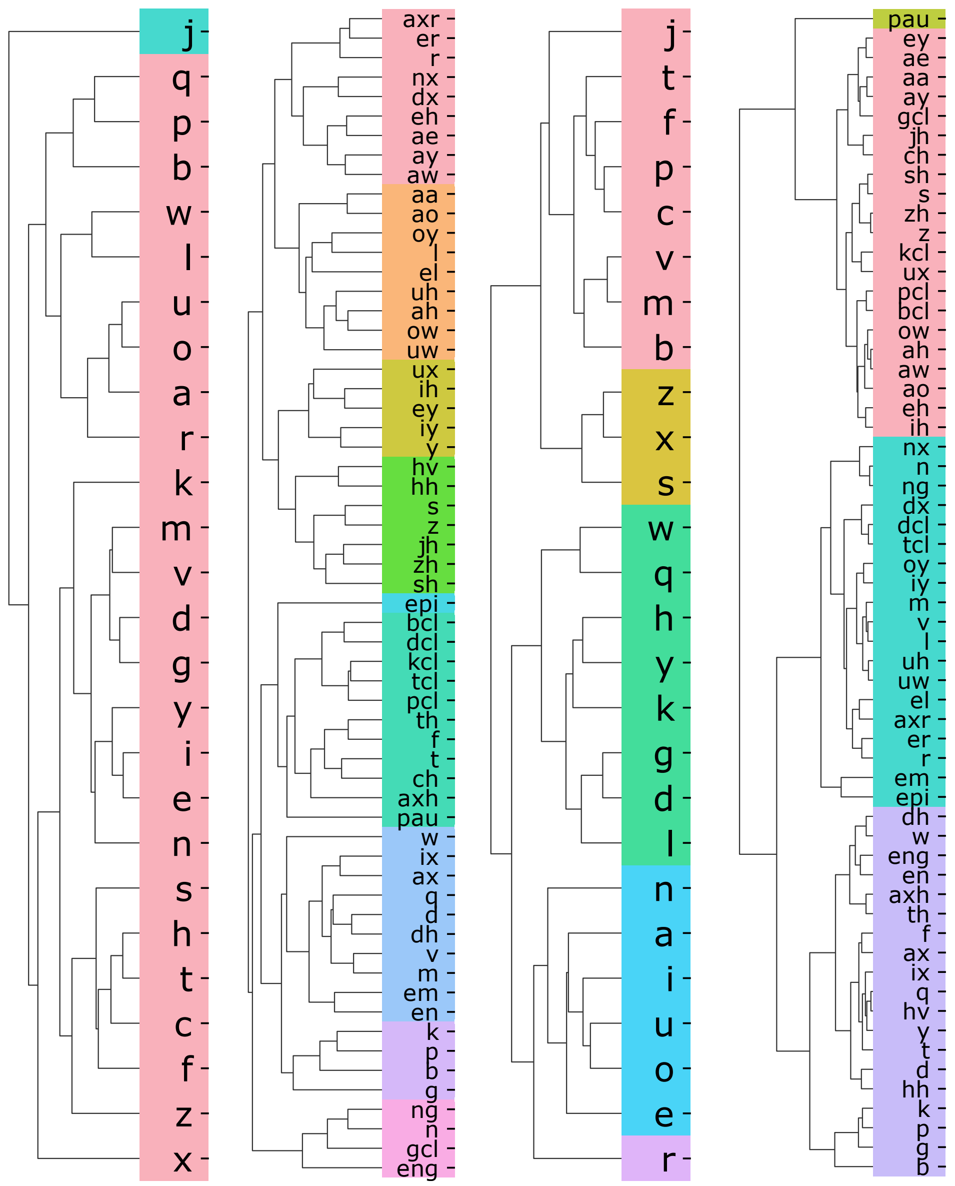


(c) letters, layer 12, 87th percentile



(d) phonemes, layer 12, 96th percentile

Figure 7.7: Clustermaps of NAPs of the W2L model in layers 5 and 12 using grouping by predicted letter and annotated phoneme. Each subfigure shows the clustering at the respective percentile of best quality according to Silhouette score. Darker shades of red indicate higher distance. Equal heat map colors represent same distance values in the same layer, but are not comparable between layers. Colors of clusters in different plots do not represent mapping of the clusters. An enlarged view of the clustering without the heat map of pairwise distances is shown in Figure 7.8.



(a) letters layer 5 96th percentile

(b) phonemes layer 5 87th percentile

(c) letters layer 12 87th percentile

(d) phonemes layer 12 96th percentile

Figure 7.8: Clustering result of NAPs of the W2L model in layers 5 and 12 using grouping by predicted letter and annotated phoneme. Each subfigure shows the clustering at the respective percentile of best clustering quality according to Silhouette score. Clustering assignment is shown by coloring. Colors of clusters in different subfigures do not represent mapping of the corresponding clusters. Pairwise distances are visualized in Figure 7.7. Overrepresented phoneme categories in clusters of (b) are shown in Figure 7.6.

the letter prediction model, in particular in the output layer. In addition, the phoneme NAP clustering result demonstrates that there cannot be a strong correspondence between phonemes and letters. If there was such correspondence, a phoneme NAP in the output layer would show high activations for a particular letter output neuron, hence show clustering behavior that is similar to the corresponding letter.

7.3 Error Detection with Topographic Maps

Topographic map visualizations can be used to identify whether classification errors are caused by wrong target annotations in the data set. Annotation errors can occur either in training data or test data which has different effects on the topographic maps. We demonstrate these effect using two toy examples, where we deliberately introduce annotation errors in either the training or test data. The topographic map visualizations are computed with the UMAP_PSO method, which we identified as the best layouting method in Chapter 6.

7.3.1 Toy Examples Design

For our toy examples, we introduce errors in the MNIST and Fashion MNIST data sets.

In the first example, we introduce a systematic error in the Fashion MNIST test data by changing the target class of 90 % of the examples of class ‘0’ (“T-shirt/Top”) to class ‘1’ (“trouser”). Using this altered test data, we create topographic maps for a model that is trained on the original Fashion MNIST training data. As Fashion MNIST is more complex than MNIST, we use the CNN model as described in Section 4.2.1.

In the second toy example, we use the MNIST data set and introduce a systematic error in its training data. Specifically, we change 90 % of the class ‘0’ examples to class ‘1’ and train a model on this altered training data set. For this model, we create topographic maps for the original MNIST test data. As architecture, we use the simple MLP described in Section 4.2.1

We use an unrealistically high amount of mislabeled examples to facilitate the understanding of the visualization. Nevertheless, both examples represent real-world scenarios that we discuss in the corresponding sections.

The groups of interest in both toy examples are the classes according to the test data annotations. Because we are further interested in the errors, we separate the examples of each class into whether they are correctly predicted by the model, resulting in 20 groups of interest. We create topographic maps for the first fully-connected layer of the MNIST MLP and the first convolutional layer of the Fashion MNIST CNN model.

7.3.2 Annotation Errors in the Test Data

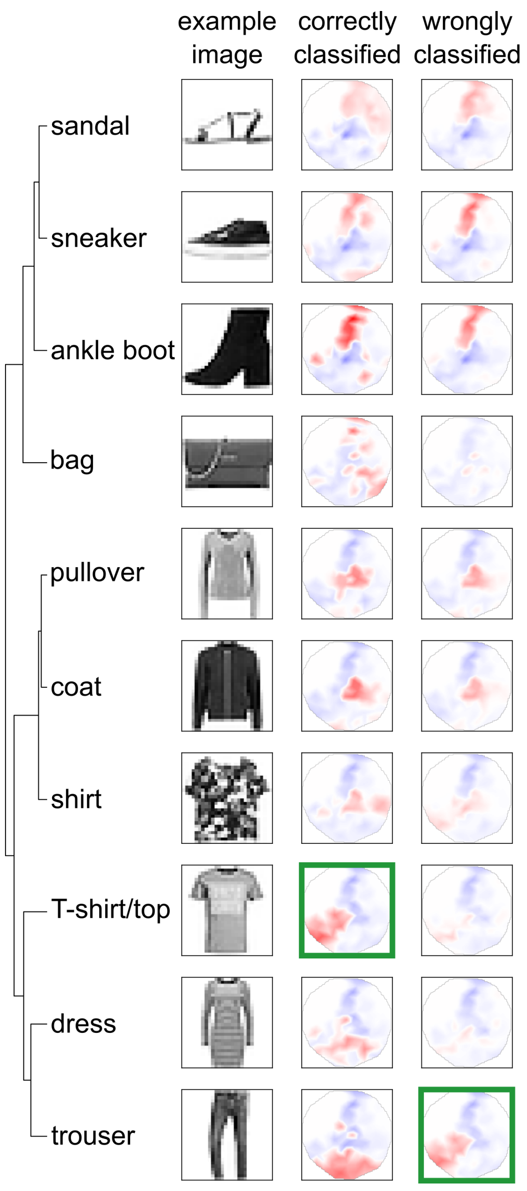
Figure 7.9a shows the topographic maps of all Fashion MNIST classes, separated into correctly and wrongly classified groups.

Two characteristics of the topographic maps indicate potential errors in the annotation of the test data. First, erroneous test data annotations lead to a high difference of the activity between the topographic maps of correctly and wrongly classified examples of the same label. Second, the activity of the wrongly classified group is similar to the activity of another group, that is, the class which the examples are supposed to be annotated as.

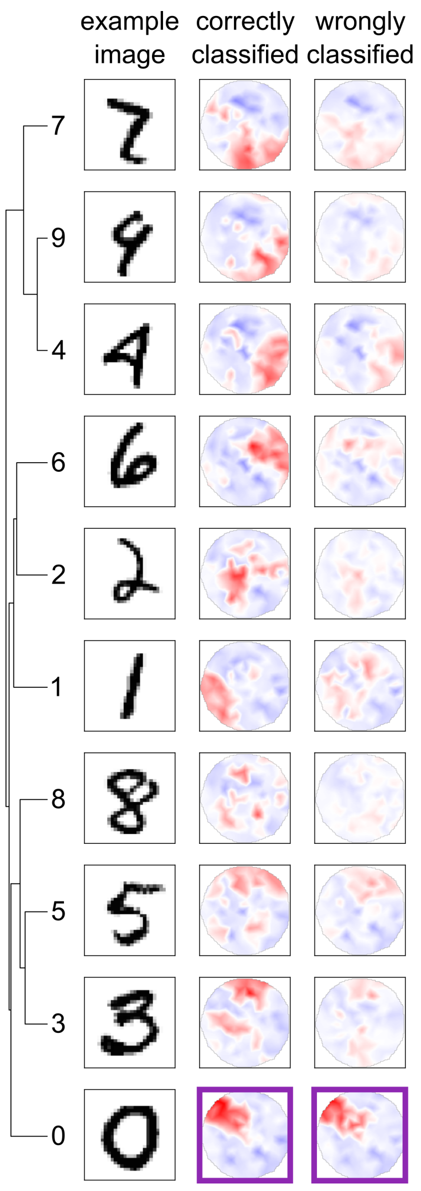
In the shown example, the first criterion is met for several classes, for example, “bag”, “T-shirt/top” and “trouser”. Only for the wrongly classified “trouser” group, we observe that the topographic map is similar to the activation of correctly classified “T-shirt/top” images (highlighted in green). This matches the error that we injected in the test data, that is, changing 90 % of the “T-shirt/top” labels to “trouser”.

In realistic models, a dissimilarity between the topographic maps of the correctly and wrongly classified groups of the same annotation indicates that there is a distribution difference in this class. If no other topographic map is similar, this low similarity can be related to using non-representative training data for this class or using test data with a highly different distribution. The model can potentially be improved by including a part of the out-of-distribution examples in the training data or by introducing a new class which represents them.

In the upper left of the topographic maps in Figure 7.9a, we observe a white region in all groups. This region is not empty because the PSO distributes the neurons in a round shape. Instead, a white region that exists in all groups corresponds to a larger subset of neurons whose activity is highly similar across the groups. This indicates that the model does not use its



(a) Test data annotation errors.



(b) Training data annotation errors.

Figure 7.9: Topographic maps for correctly and wrongly classified examples using data sets with annotation errors. (a) Annotation errors in the test data. (b) Training data with annotation errors. The activation similarity is shown as a dendrogram and used to sort the classes. The shown example images are randomly chosen from the respective group. The green and purple annotations highlight the pairs of topographic maps that indicate the error in the respective example.

full capacity, for example, because it is too complex for the given task or due to training problems like ReLUs that never activate [109]. Considering that the used Fashion MNIST CNN only has a train accuracy of 84.79 % and a test accuracy of 83.59 %, overcomplexity is an unlikely reason for this model.

7.3.3 Annotation Errors in the Training Data

To investigate training data annotation errors, we train a model using an erroneous MNIST data set. The resulting topographic maps of correctly and wrongly classified groups when evaluating with the original MNIST test data set are shown in Figure 7.9b.

Like in the previous example, we visually compare the topographic maps to identify potential errors in the training data. However, the criteria are different from the ones for the test data. Annotation errors in the training data lead to a high similarity of topographic maps for wrongly and correctly classified examples of the same group. This means that the model detects similar patterns in both groups but still categorizes the examples differently. In addition, there is typically no other topographic map that is highly similar to the potential error candidate.

We observe this pattern for class ‘0’ (purple highlight), which again matches our injected annotation error of changing 90% of the ‘0’ labels. However, using the binary split into correctly and wrongly classified examples does not show which class the training examples are mislabeled as. To identify the class of the wrong annotations, we can extend the analysis by creating a confusion matrix-like topographic map visualization. There, we can find that the topographic map at the ‘0’-classified-as-‘1’ position is highly similar to the correctly classified ‘0’ examples (see purple highlight in Figure 7.10).

Observing similar topographic maps of the correctly and wrongly classified examples of the same class can also give insight in realistic models. Commonly, this pattern occurs if the model cannot discriminate between two or more classes properly. In this case, there are multiple classes that share similar topographic maps in the correctly and wrongly classified groups. One example can be found in the classes “sneaker” and “ankle boot” of Figure 7.9a. There are no changes in the annotations of these two classes but the topographic maps of the corresponding groups look similar. This

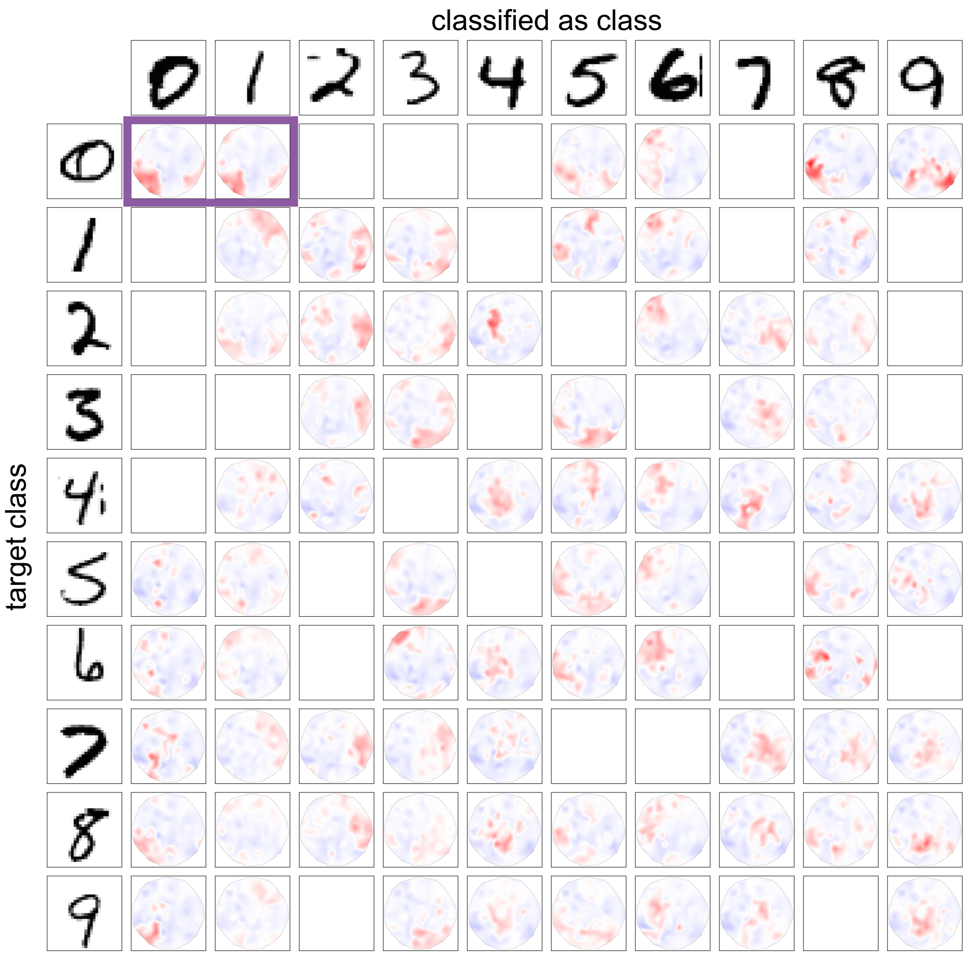


Figure 7.10: Topographic maps for correctly and wrongly classified examples, with the misclassification groups further split by the predicted label. The examples are based on a toy example model trained on data with annotation errors. The digit images are exemplary inputs of the respective group. Empty topographic maps indicate that the corresponding wrong classification was not made by the model. The purple box highlights the pair of topographic maps corresponding to the introduced data annotation error.

indicates that the classes are too similar to each other for the model to discriminate them well. Improvements of the model can be achieved by merging the classes that are similar to each other or by increasing the model capacity to strengthen its ability to distinguish similar classes better.

7.4 Detecting Bias with Topographic Maps

DNNs are prone to reproducing or emphasizing biases that are contained in the data used to train them. Different approaches to detect and mitigate bias have been introduced. For example, Sweeney [176] shows racial discrimination in the online ad delivery by Google and Bolukbasi et al. [13] debias commonly used word embeddings. More recently, researchers investigate biases in modern transformer-based models like BERT [30], for example focusing on gender bias [100] or ethnic bias [5]. Discrimination in facial recognition algorithms is evaluated by Buolamwini and Gebru [15] and they introduced the Gender Shades data set with a balanced distribution of gender and skin type. Similarly, Karkkainen and Joo [68] introduced the Fair Face data set which is balanced for gender, race and age categories. In this section, we perform racial bias detection as an exemplary use case of our novel visualization technique.

Different to Buolamwini and Gebru [15] or Karkkainen and Joo [68], who investigate bias in the output and the performance of algorithmic decision systems, we focus on the representations of the individual layers of a DNN. We demonstrate an exemplary application of detecting representational bias in a pre-trained model. Mitigating this bias, however, is not within the scope of this thesis.

7.4.1 Experimental Setup

As a real-world example, we investigate the bias in the representations in VGG16 [170], which is a pre-trained CNN model that can be used as feature extractor for downstream applications like image recognition DNNs. We investigate the representations of a pre-trained VGG16 model, obtained from the TensorFlow Keras applications[1] module, using the second and fifth

[1] https://github.com/keras-team/keras

maxpooling layer (layers 6 and 18) as an example. The VGG16 architecture is shown earlier in Table 4.6.

As test data, we use FairFace [68], a balanced data set of images of people from different age groups, races and binary genders. Here, we choose the “race”, “gender” and “age” variables as different groupings to compute the topographic maps, using 500 randomly drawn examples from each group. Moreover, as a random baseline to compare with, we add several groups of randomly picked examples to compare whether the topographic maps of the sensitive variables are easier to distinguish from each other than the random groups.

Topographic maps for sensitive variables in layers 6 and 18 of VGG16 are shown in Figures 7.11 and 7.12. In the appendix, we provide topographic maps of multiple layers of the network (Appendix Figure A.12).

7.4.2 Results – Sensitive Variables

“race” In layer 6 (see Figure 7.11 left), each “race” category has a specific activation pattern. We only observe a small common active region for Black and Indian categories. However, in layer 18, class activations become more similar between particular groups. Specifically, Black and Indian categories are highly similar, as well as East Asian, Southeast Asian and Latino Hispanic. The observations indicate that there is a racial bias in deeper layers and poses the risk that downstream applications perform decision on the encoded race similarities. Surprisingly, the Middle Eastern and White categories do not show clear activation patterns, potentially because the model learns more individual representations for these categories.

“gender” Topographic activation maps of categories Female and Male (Figure 7.11 right) are almost the inverse of each other, which is expected for a binary grouping. With only two variables to compare, we focus on the comparison to the random groups. We observe that, in layer 6, there is little difference between random groups and sensitive variables in how strong the activation deviates from the global mean. In layer 18, this difference becomes larger. Clearly, in layer 18, the Female and Male groups show stronger over-/under-activation than the random groups, indicating that the representation in deeper layers distinguishes between these categories.

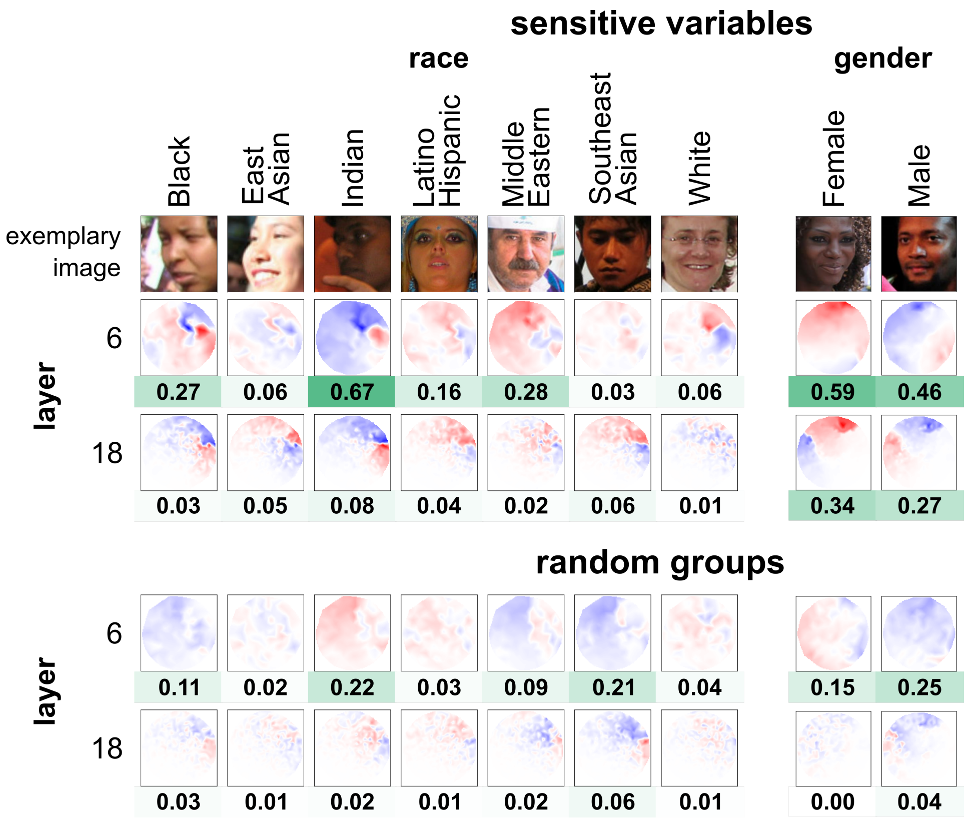


Figure 7.11: Topographic activation maps when grouping by “race” and “gender” variables in layers 6 and 18 of VGG16 and when grouping randomly. Size-weighted convexity is shown below each topographic map. For “race” and “gender” respectively, topographic maps in the same layer use a common color map.

“age” Groups of similar age (Figure 7.12) are similarly activated, which is reasonable considering the categories’ fuzzy boundaries. In layer 6, we observe clusters of high similarity: age groups 0-2, 10-49 and >50. Layer 18 shows more continuous changes with strongest activation differences from the global mean in the lowest and highest age groups. This indicates that the representations encode age information without a systematic disadvantage for any group. Nevertheless, a downstream application would be able to learn biased decisions.

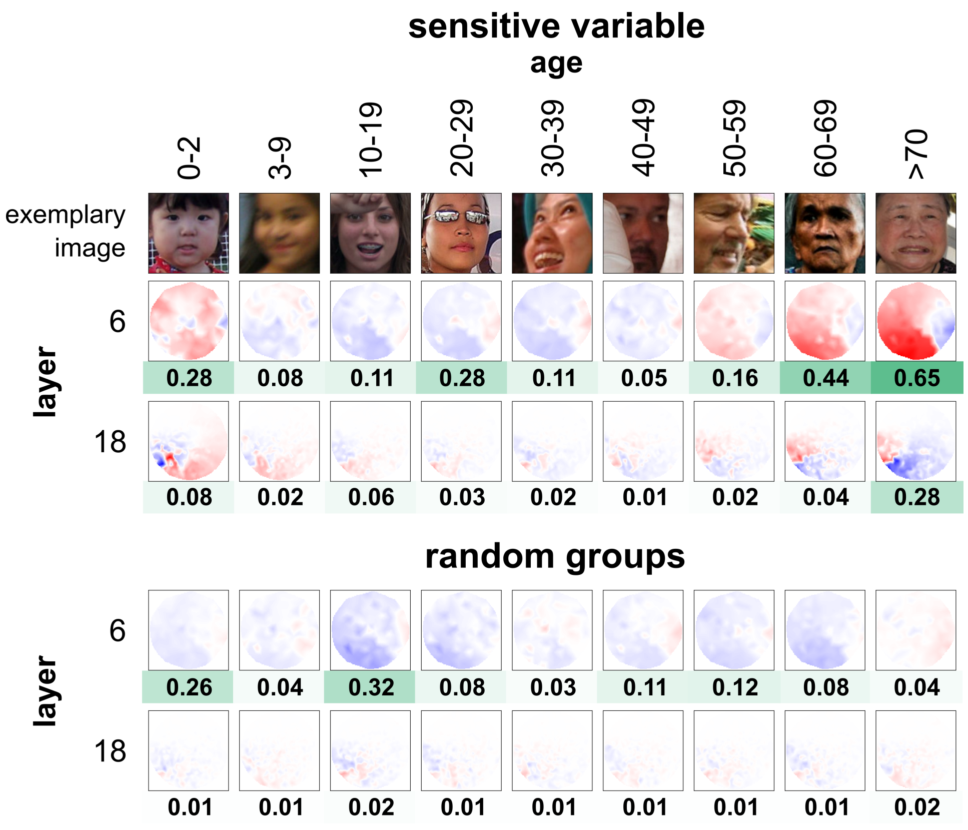


Figure 7.12: Topographic activation maps when grouping by "age" variable in layers 6 and 18 of VGG16 and when grouping randomly. Size-weighted convexity is shown below each topographic map. Topographic maps in the same layer use a common color map.

7.4.3 Results – Significance

In Figures 7.11 and 7.12, qualitatively, we observe stronger color intensity for sensitive variables compared to random groups. This stronger deviation from the global mean activation indicates that the observed activation patterns are significant and not only a random effect. Further, we observe more pronounced activation differences between the sensitive variables compared to between the random groups, which indicates that differences are really related to the sensitive variables. Both observations are more clear in deeper layers, which hints that distinction between the sensitive variables becomes stronger the more complex the learned patterns get.

Quantitatively, we notice that visual quality according to size-weighted convexity is affected by the number of feature maps. Fewer feature maps might lead to higher visual quality because the layout can be easier optimized for them. In our example, fewer feature maps are active simultaneously leading to smaller highly-active regions and larger white regions, that are, group-unspecific feature maps. This decreases the visual quality. Therefore, layer 18, which is wider than layer 6, shows lower visual quality.. The number of groups, however, is not a critical factor. In general, the size-weighted convexity of visualizations for sensitive variables is higher than for the corresponding random groups which supports the significance of the results.

7.4.4 Diversity of Groups

We appreciate the efforts of the FairFace data set to provide balanced evaluation data. However, the values of the provided sensitive variables still have potential to be further diversified. For example, “gender” is only considered as a binary or certain age groups, like “more than 70”, include a larger range of ages than other groups. We still consider the data set as suitable to demonstrate our technique but encourage the research community to conduct studies with more diverse data sets upon availability. In this work, we only consider the sensitive variables independently. As this does not consider intersectionality, we also encourage investigating combinations of sensitive variables in future work.

8 Extensions

In this chapter, we discuss several ways of extending our methodology. This involves preliminary work as well as discussion of possible future research directions.

8.1 Multi-Layer Visualization

In the previous chapters, we showed NAP analyses and topographic activation maps mainly for individual layers. However, for a more comprehensive overview of the network, our method can be extended to multiple layers, as well.

The straightforward approach is to simply perform the same type of analysis for several individual layers. One example can be found in Section 7.2, where we show representational similarity plots of the ASR models in different layers. Another example is Appendix Figure A.12, which shows topographic maps in different layers related to the bias representation experiment of Section 7.4. As there typically is a large amount of layers in DNNs, it is infeasible to show all layers at once. We recommend to carefully choose a representative subset of layers in different depths of the network (compare Sections 5.1.1 and 7.1.1) This approach on visualizing multiple layers, however, does not relate the layers to each other but only shows the individual layer results next to each other.

A more advanced approach for achieving a multi-layer analysis is to determine relations between the layers. For topographic maps, a visual way of relating layers is to align the layouts with Procrustes superimposition, that is, translating, rotating and uniformly scaling the topographic maps of one layer to optimally match those of another layer. In our case, as we already scale topographic map layouts to a common size, Procrustes superimposition reduces to finding the rotation that optimally matches the topographic maps in two layers, according to the minimum MSE across all groups.

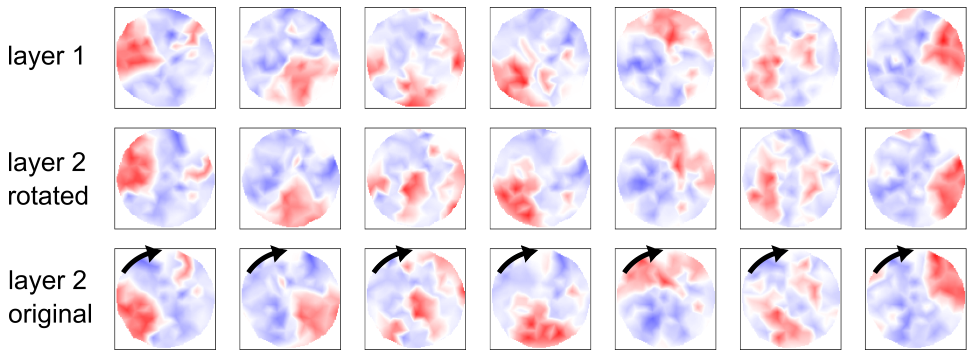


Figure 8.1: Topographic maps of multiple groups in two layers of a DNN. The original topographic map layout of layer 2 (bottom) is rotated to best match to the layer of layer 1 (top), resulting in the layout shown in the middle row.

This approach follows the assumption that information gets forwarded in the network through neurons or feature maps of similar activation. However, this is not generally true. For example, consider a neuron of high activation in a layer. This neuron does not functionally relate to a highly activated neuron in the following layer if their connecting weight is small. On the other hand, a lowly active neuron can be connected with a high negative weight, making it strongly influenced by this particular input neuron.

To also capture functional relations between the regions, it is necessary to also analyze the weights that connect the corresponding neurons or feature maps. However, this is a highly-complex problem, both from an analytical and a visualization perspective. Firstly, there are relations between any pair of neurons or feature maps from two layers, leading to a large number of pairs to evaluate. Secondly, while the connection of two neurons in fully-connected layers is a single value in the corresponding weight matrix, the connections in convolutional layers are more complex as they apply the weights as convolution kernels. Finally, even if we can obtain representative correlation values, including multi-layer constraints complicates the layout optimization immensely as it requires multiple optimization criteria that potentially counteract each other. We already made a similar observation of conflicting optimization targets in our PSO layouting approach. The PSO algorithm (compare Section 6.1.2) applies attracting forces to similarly active neurons and repelling forces to dissimilar neurons. Due to the high number of neuron pairs that influence each other simultaneously, the resulting layout cannot find a satisfying optimum in many cases. Including additional factors to this layouting algorithm to encourage functional relation of similarly located positions would likely amplify this problem.

8.2 Training Processes

In the main thesis, we analyze DNNs at the end of their training. Additionally, it is interesting to perform this analysis for the model at different time steps during training. This can reveal information about the training process of a DNN. For example, it can indicate that some classes are learned in earlier training stages than others, or that a particular type of error only occurs after a certain time in training.

8.2.1 Visualization Oriented at the Final Layout

One approach is to orient the layout at the final model state in order to visualize how the regions in the trained model develop over time. For the fully-trained model, we compute a topographic map layout using UMAP_PSO and use it for all other training steps. This way, the layout stays the same and we observe the development of activations during the training.

As an exemplary model training, we train a simple MLP model on MNIST for one epoch and save the model after each batch resulting in a total of 1,875 model states. Figure 8.2 shows this training process as topographic maps at exemplary time steps. We observe, that the change of DNN activations is particularly large in the first 100 batches and gradually changes less as training continues. Correspondingly, we show a smaller step difference early in training than towards the end of the epoch. We observe that initial shapes begin to emerge after approximately 20 batches. These shapes become more sophisticated after around 120 batches and continue to become clearer over time. After around 500 batches, we only observe minimal changes. Further, we observe that the topographic maps of some classes stabilize earlier than others, which indicates that they are easier to learn for the model. For example, classes ‘1’ and ‘4’ change little from batch 500 to the end of the epoch, while some active regions disappear for

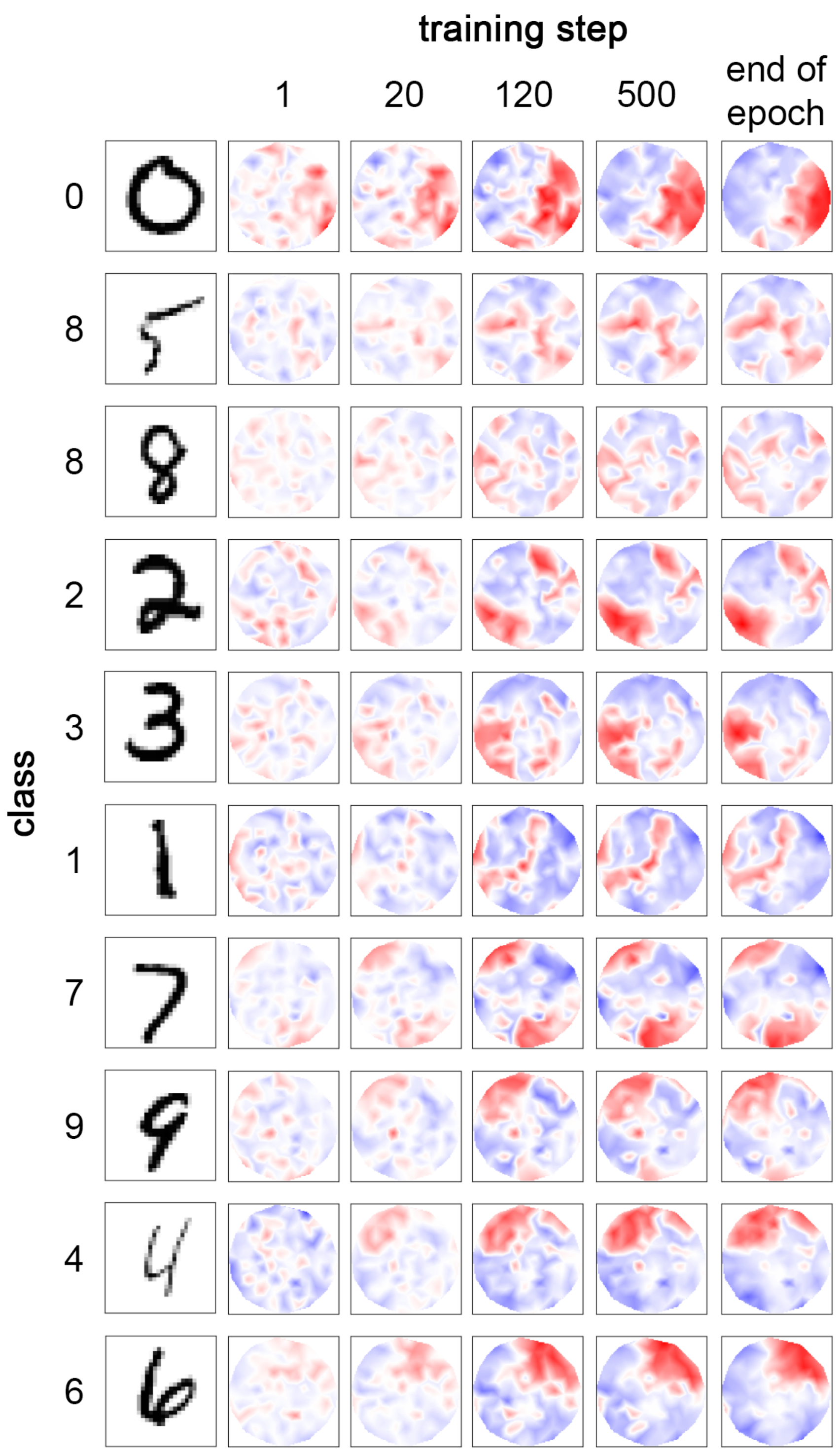


Figure 8.2: Topographic maps at different stages of training. Classes are ordered by activation similarity.

classes ‘0’ and “6”. Li et al. [101] found different classes to be learned early and late by their used CNN model, indicating that these learning dynamics are model-dependent.

Topographic maps provide a unique perspective on the training process and help to better understand the dynamics of the neural network training. However, due to processing many model states, the analysis of training processes requires more computation than for a trained model only. Additionally, in this approach that is oriented at the final layout, all model states have to be saved before creating the topographic maps at the respective states. Storing the same model for every training step requires a lot of storage capacities, as well. For example, the 1,875 saved model states in the shown exemplary MLP trained on MNIST summed up to around 2.4 GB.

8.2.2 Temporal Resolution

In the previous example, we picked exemplary time steps manually based on how strong the activation changes between them. However, in practice it is infeasible to manually inspect all training steps to define the steps for the visualization. A future extension of training visualization is therefore to determine an automated process of choosing suitable model states. Generally, we expect to require more steps in earlier stages of training and less when the model is getting closer to its optimum. However, determining a generally applicable rule to describe optimal step sizes over training time is highly challenging. A potential research direction is to use changes in loss or performance metric as a proxy to determine steps to visualize. However, the loss is not necessarily correlated with the visual quality change of topographic maps. Moreover, in many training processes, there are, for example, plateau phases or temporary decreases in performance which complicate applying a simple rule to determine suitable steps.

8.2.3 Adapting the Layout over Training Time

Orienting the visualization only on the final layout has the disadvantage that it follows the assumption that the final function of a neuron is already pre-determined in the beginning of the training. However, in reality, neurons might change their functionality over training time. Therefore, different neurons can co-activate early in training than later. Computing a separate layout for every training step, however, is not useful because the individual visualizations will not align and make comparisons impossible. Therefore, it would be necessary to dynamically optimize the layout during training. This means, starting from a (random) initial layout, the layout coordinates are optimized for the first step. All succeeding steps will use the layout from the previous step and only update it slightly to account for the changes during the respective training step. Such continuous optimization in parallel to training is, however, very challenging to balance. If the updates are too small, the layout might not represent the step correctly and if they are too large, it might lead to very different and incomparable layouts between subsequent steps.

8.3 Relation of Confidence and Activations

In this application example, we investigate how network activations relates to its confidence. To this end, we group the data by predicted label and the model's confidence and visually compare activations for all groups at the end of the model training and during the training process. Further, we investigate the relation between activations and overconfidence of the model, that is, when the confidence is higher than the accuracy. With this experiment, we like to demonstrate the possibility of using informed subgrouping (Section 5.1.7), and combine it with training process analysis.

Evaluating confidence of DNNs can be done, for example, by the softmax output of a model, using deep ensembles or calibrating uncertainty after model training using temperature scaling [145, 183, 118, 150]. Here, we focus on activation visualization. With this scope, we decide to use softmax output confidence, knowing that it is not the most reliable measure [145].

8.3.1 Experimental Setup

We use the MLP architecture trained on MNIST as described in Section 4.2.1. We perform experiments with different groupings. As an overview of class activations over training, we group the data set by the predicted label, resulting in 10 groups. To investigate the relation of activations and confidence, we group by class, softmax output value bins $[0, 0.1], (0.1, 0.2], \ldots, (0.9, 1.0]$ and whether the prediction is correct, leading to $10 \cdot 10 \cdot 2 = 200$ potential groups. Confidence bins $[0, 0.1], (0.1, 0.2]$ do not contain elements for our model so we omit 40 groups. Further, we only report results for correct predictions because there are only few erroneous predictions for MNIST. To compare experiments, we use the same layout and color map for every topographic map, obtained from the last training step using the confidence- and class-separated groups.

8.3.2 Results

Activations over Training Time

Figure 8.3 shows the training of a model on MNIST as topographic activation maps at exemplary training steps. Changes in parameters, and consequently changes in activations, are larger in earlier stages of training. Therefore, the difference of the chosen steps is larger for later training stages.

We clearly observe that the activations gets stronger during training and saturates when the model is fully trained (high color intensity and contrast). Moreover, over- and under-activated neurons (red and blue) only form clear regions later in training. This seems unsurprising considering that we use the final layout as a reference, especially for the initial step. However, we still observe changes of regions in late training stages. This indicates that

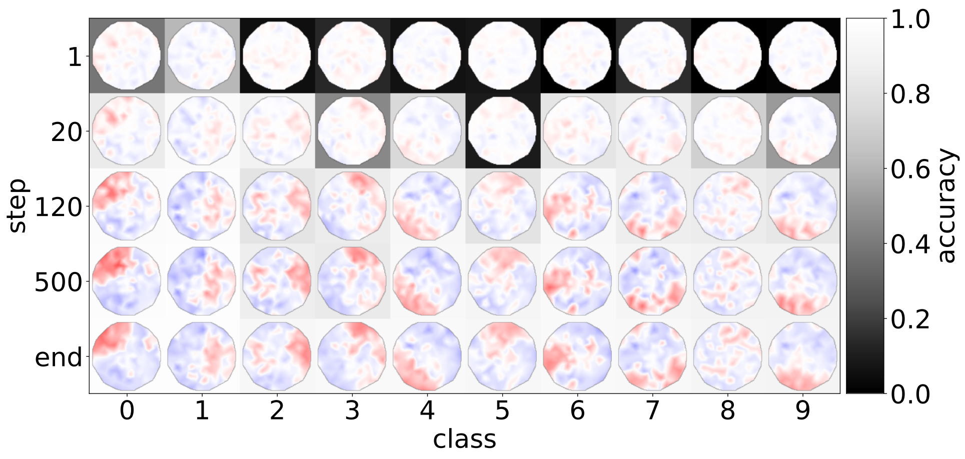


Figure 8.3: Activations evolving over training time, shown as topographic maps at different stages of training for different predicted labels.

neurons still change their function instead of just becoming more specific to a certain stimulus over training time.

From the final training step in Figure 8.2, we can compare similarity of classes according to the model activations. We observe similarities between labels ‘4’ and ‘9’ and partly overlapping active regions, for example, for classes ‘1’ and ‘7’ or ‘2’ and ‘3’. Subjectively, this fits our expectation of how classes are visually similar. Some classes do not show clear regions, for example, ‘8’. This is partly due to using the layout computed from the confidence-separated groups instead of optimizing the layout for the 10 classes.

Activations and Confidence

Figure 8.4 shows topographic activation maps that contrast activations for different labels (left) and for class ‘0’ at different training steps (right) at different confidence levels. The heat map in the background indicates the prediction accuracy for the corresponding group. Groups with purple outline are overconfident, that means, the confidence is higher than

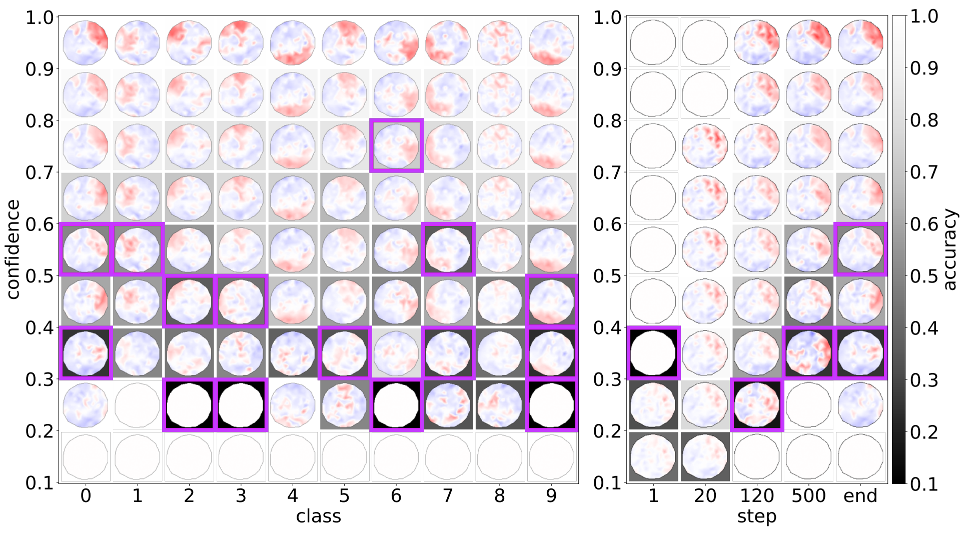


Figure 8.4: Topographic maps at different prediction confidences for different predicted labels (left) and for label ‘0’ at different stages in training (right). Background heat maps show accuracy and a purple outline indicate overconfidence. Positions with all-white topographic maps are empty groups or groups without correct predictions.

the accuracy. As we use confidence bins, we compare accuracy to the respective bin's center value. An overview of all time steps can be found in Appendix A.6.

Generally, we observe that activation strength increases with the confidence of the prediction. There are exceptions like label '7' in confidence bin $(0.2, 0.3]$ that are due to a small number of examples in the group and hence smaller averaging effects. As expected, predictions become more accurate and confident over training time. We almost only observe overconfidence in the low confidence regions, which is reasonable. An exception is a weak overconfidence for class '6', however only with confidence of 0.7 to 0.8 at an accuracy of 0.72. Further, we observe that active regions mostly form for accurate and confident groups while overconfident groups show different or scattered patterns. It is noteworthy that we observe this although the layout is optimized over all groups (in the last training step).

8.3.3 Discussion

Combining DNN activation visualization with confidence analysis can give insights into how confidence and activations relate and how they develop over training time. For demonstrative purposes, we only presented a very simple model and data set. Nevertheless, the methodology can be applied to other types of DNNs, larger data sets and more complex data types. However, the results might be more difficult to interpret, for example, because the visualization becomes cluttered for larger numbers of classes or deep models would require to investigate activations in multiple layers. In future work, we will research these limits and improvements to make it better interpretable for large-scale models. Furthermore, we hypothesize that longer training leads to higher confidence while not increasing the accuracy in the same amount, leading to more overconfidence. Testing this hypothesis will be subject of future work as well.

8.4 Downstream Explanations

NAPs and their visualization as topographic activation maps provide a high-level overview of activations and how they compare between groups of inputs. For more detailed explanations, there is the possibility to use our technique to apply other explanation techniques in a more targeted way. For instance, NAPs value zero indicates neurons that are unspecific to the groups of interest and are potential candidates for targeted network pruning, that is, removing irrelevant neurons from the DNN. As another example, NAPs can be used to identify sets of neurons that are highly or lowly active for certain groups. These sets can then be used as targets for feature visualization (Section 3.4) or to create saliency maps for these neurons (Section 3.5). This way, it is possible to visualize patterns that are learned jointly by multiple responsive neurons. Using topographic activation maps, it is further possible to select target neurons interactively. Figure 8.5 shows a user interface that we currently develop to interact with topographic activation maps, where regions can be selected for which a feature visualization can be performed. Regions of neurons can be selected manually, but we can also use the pre-processing steps of the connected component quality measures from Section 6.2.1 to automatically select, for example, the largest active region.

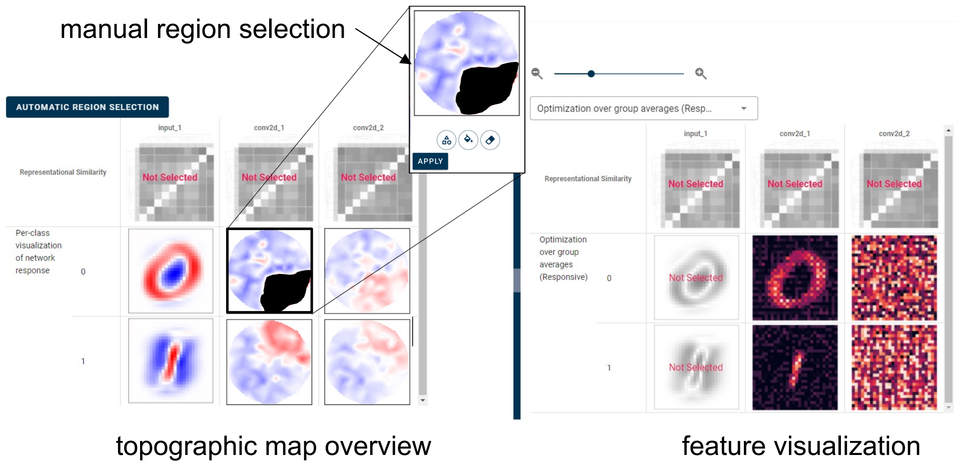

manual region selection
AUTOMATIC REGION SELECTION
input_1
conv2d_1
conv2d_2
Representational Similarity
Not Selected
Per-class visualization of network response
0
1
APPLY
Optimization over group averages (Resp...
Optimization over group averages (Responsive)
topographic map overview
feature visualization

Figure 8.5: Targeted feature visualization using topographic maps. In a graphical user interface, regions of neurons can be selected (black selection on top of the topographic map) for which their jointly learned pattern can be visualized (on the right side).

9 Conclusion

In this thesis, we introduced a novel neuroscience-inspired approach on analyzing and visualizing DNNs activations. As two main components of this approach, we presented Neuron Activation Profiles (NAPs) for characterizing DNN activity for groups of inputs and topographic activation maps as a visualization of NAPs. We evaluated both methods in detail and provided several application examples and extensions of the methodology. In particular, we included applications to Automatic Speech Recognition (ASR) to demonstrate the applicability to data that are not visually interpretable.

To conclude the thesis, we will first discuss whether we were able to meet our research aims, including stating the main findings for each aim. Further, we will critically evaluate how closely our approaches resemble their counterpart from neuroscience and whether using the neuroscience-inspired technique is advantageous. Finally, we will discuss promising future research directions based on our findings.

9.1 Research Aims

We followed three research aims (compare Section 1.1). In this section, we discuss to which extent we were successful in proposing solutions to these aims and whether there is potential for improvement. Further, we discuss how our experiments address topics in addition to the respective research aims.

9.1.1 RA1: Group-Specific Network Responses

To "Characterize group-specific network responses with minimal information loss from aggregation", we introduced NAPs (Chapter 5). While NAPs are generally not interpretable themselves, we found them to be useful for a comparative analysis of groups regarding the network activations. This particular usefulness for comparisons is further amplified by visualizing NAPs as topographic activation maps (RA2).

Alignment is a crucial aspect of our averaging approach. To not lose information when performing the averaging over activations of multiple examples, we investigated an alignment based on highest saliency. Our evaluation showed that our saliency-based alignment is suitable to obtain useful averaging results, in particular for well-generalized models (Section 5.3). Nevertheless, NAP computation is best applicable for data that are aligned based on knowledge about their features.

Another critical aspect for NAPs to yield useful representations of network activations for a certain group is the choice of the group itself. Averaging the activations removes variation coming from the data within the group. If the group, however, has a too high variance and few common features, the averaging approach cannot determine representative activation patterns (Section 5.3.1). As of now, we are not able to suggest a measure to evaluate whether a group is suitable for computing a NAP. However, we recommend to only choose groups that are expected to share characteristics.

Our evaluation of NAPs for understanding representational similarity in a DNN included a manually constructed example of ASR models where we have an expectation about in which layers particular groups are represented (Section 7.2). While the results are very promising, a general evaluation of the approach is difficult because there are no benchmark applications available that provide a ground truth representational similarity to compare against.

While developing the approach, we found that it requires much computational effort due to computing averages over many examples and the large dimensionality of activations. Therefore, we further investigated how to improve the efficiency by only using random subsets of groups (Section 5.4) which was not part of the original aim. Approximating NAPs is possible for groups with much redundancy, however, the difficulty lies in determining how many examples are required to approximate a NAP.

9.1.2 RA2: Summarization through Visualization

To “Summarize complex representations through visualization”, we developed a visualization of NAPs as topographic maps (Chapter 6). They leverage the comparative power of NAPs by providing a means to not only display the similarities but to also visualize the values themselves. This way, topographic activation maps allow to visually compare group activations and additionally show which sets of neurons activate similarly and differently between the groups.

We investigated various methods to compute a topographic layout (Section 6.1.2) using different measures of visual quality (Section 6.2.1). Our results showed that, among the considered layouting techniques for topographic maps, UMAP_PSO is the most generally applicable (Section 6.3.3). UMAP_PSO first computes a two-dimensional Uniform Manifold Approximation and Projection (UMAP) projection and distributes the points evenly with a local force-only Particle Swarm Optimization (PSO). Further, our introduced size-weighted convexity measure is most suitable for quantifying the visual quality of the visualization (Section 6.3.2). In CNNs, feature maps need to be aggregated in order to assign them a single representative NAP value for visualization in the topographic map. We found that both mean and center value produce similar results while maximum value aggregation led to uninformative topographic maps. To be generally applicable, we find mean aggregation to be most suitable (Section 6.3.3).

Topographic activation maps are similar to dimensionality reduction techniques. However, they focus on the ease of visual inspection and comparison and the possibility of visually estimating regions and their sizes. This requires the distribution using our local-force PSO. Compared to other dimensionality reduction techniques like tSNE and UMAP, topographic maps therefore omit information about pairwise similarities. Consequently, our visualization cannot be used for drawing conclusions about neuron similarities but only about activation similarities between groups.

We demonstrated a variety of applications using topographic activation maps. This included error analysis, bias detection, visualizing training processes and relations of model confidence and activations (Sections 7.3, 7.4, 8.2 and 8.3, respectively). Through these applications, we found that topographic maps are a very versatile visualization technique for DNN responses. In the model confidence experiment, we further demonstrated that they can be combined with other common visual analyses, in this case with an accuracy heat map.

We found topographic maps to be a successful solution for RA2, both subjectively and according to our quality measures. Mostly, we focused on the aspect of visual quality. Still, we considered the faithfulness of the visualization. To this end, we used toy examples in which we expected certain topographic map similarities (error detection in Section 7.3.3) and performed significance analyses to show that information in topographic maps is not only a random effect (in Section 7.4). In both examples, we found that topographic maps are not only visually appealing but also show representative information about the groups.

The usefulness of visualizations is also tied to whether people can get information from them. In our research aim, we did not specifically aim to investigate this aspect. However, we gathered unstructured feedback at several scientific conferences, trade fairs and through science communication activities. This way, we got opinions about our visualization from scientists from different research fields, users from industry and interested lay people, which covered a wide range of levels of expertise. Generally, we found that topographic maps are conceptually and visually well perceived by people across many target groups. However, using them to gain insight into DNN requires practice and we found that when seeing topographic maps the first time, an expert is required to guide people how to interpret the results in detail. Consequently, our visualization technique has potential to be improved regarding its intuitiveness.

9.1.3 RA3: Applicability to Visually Uninterpretable Data

We aimed to "provide analyses and visualizations that do not rely on visually interpretable data". To this end, we particularly evaluated our technique and demonstrated applications using W2L-based ASR models (Sections 5.3 and 7.2). We found that there is no difference in interpretability of our analysis comparing image data and speech data (as spectrograms). This is due to analyzing similarities between NAPs instead of projecting information in the input space. Therefore, we conclude that we successfully developed a model analysis technique that does not rely on visually interpretable data.

Note that we did not present topographic map visualization for the ASR models because of the large number of groups that we investigated. For example, displaying topographic maps for 62 phonemes along with the clustering results in Section 7.2 would not have supported comparing the groups because of the large number of visualizations. Moreover, there are several subsets of (typically similar) phonemes whose topographic maps are almost indistinguishable. Nevertheless, creating the visualization is possible for letter and phoneme groups, too.

While we only showed applications for images and speech, our technique is generally applicable to any DNN because activations can be obtained and compared for each model. The interpretability of the results, as discussed, depends on the specific properties of the DNN and the groups of interest.

9.2 Neuroscience Inspiration

We stated that our methods are inspired by neuroscience. In this section, we discuss how related the final techniques really are to their neuroscience counterparts. Generally, we like to emphasize that the connection between common DNNs and brains is more of narrative nature and less based on actual equivalent components. Our inspiration primarily arose from neuroscience's capabilities of investigating complex opaque system but for communicative purposes, we also appreciate to be able to draw the narrative connection from DNNs to the brain.

9.2.1 NAPs as an ERP-Like Analysis

NAPs as an activation averaging approach with alignment is similar to ERP analysis in EEG. Both are ways of identifying (artificial) neural activity to a certain stimulus or type of input. However, it is not exclusive to the field of neuroscience. Averaging approaches are frequently used to obtain information that are common to a set of examples, for example, for noise reduction in images [42]. Similar to ERPs, to obtain proper averaging results, the examples to average over need to be aligned. In computer vision this alignment is commonly called image registration [46, 29, 78]. Therefore, NAPs are not exclusively a neuroscience-inspired technique but have similarities to other aligned averaging approaches as well. Nevertheless, focusing on the connection to ERPs allows a logical transition to topographic activation maps because both are applied in EEG data analyses.

9.2.2 Topographic Activation Maps

Different to NAPs, our visualization as topographic map is closely connected to neuroscience. We specifically aim to achieve an activity visualization for DNNs that is visually similar to how brain activity is displayed in EEG data analysis. Generally, projecting high-dimensional data to a two-dimensional space is very common. However, the equal distribution in a round shape and the interpolation to obtain a continuous coloring is specifically related to the neuroscience inspiration. Arguably, from an information visualization perspective, there are more representative techniques. For example, as discussed, the information about pairwise similarity of neurons gets lost by avoiding gaps between them in the two-dimensional layout. However, due to the well-known visualization of brain activity as topographic maps in functional magnetic resonance imaging (fMRI) or EEG, more people can associate our topographic maps with the neuroscience counterpart which can support the understanding of the visualization.

Interestingly, the visual similarity to topographic maps in neuroscience can be detrimental for interpretability if a person has expertise in neuroscience. During our research, we noticed that neuroscientists (and researchers from related fields) tend to connect real brain areas to the topographic maps of DNNs. However, although there is a visual resemblance, there is no functional relation at all. Therefore, we found that the visualization is more intuitive for people who only have a high-level idea of brain activity, but tends to be misleading for domain experts from neuroscience.

9.3 Future Work

We see promising future research directions in three categories: stronger user orientation, technical improvement and methodological extensions.

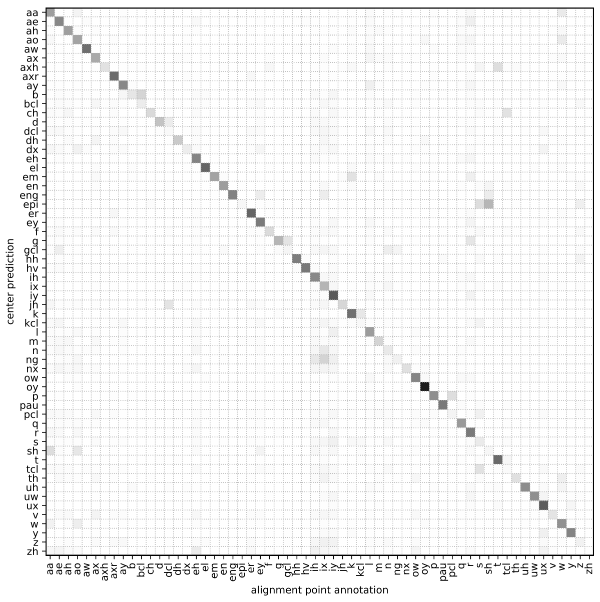
User Orientation Currently, our approach provides insight into the DNN but no explicit explanations of the model behavior. To obtain a more user-friendly experience, there needs to be better automated guidance on how to interpret the results of our approach. Furthermore, people usually prefer explanations that allow to take actions. Therefore, our technique would benefit from investigating how it can be used to improve models or to use the results to obtain quality criteria for the DNN. Future work in these directions would also require to incorporate more structured human feedback in the development process. Especially, the aspects of intuitive interpretability and actionability involve structured user studies. Moreover, these studies should address people from various backgrounds because background knowledge and personal experiences can strongly affect how the results are perceived.

Technical Improvements The most critical technical improvements are related to efficiency and scalability. Methods that analyze entire data sets are inherently costly in terms of compute resources and can also require substantial storage capacity if intermediate results need to be saved. In our approach, this particularly affects the NAP computations, which align all data and activations and perform averages over them. To reduce computation time and storage requirements, future work can investigate approximations and more efficient implementations. We showed initial results on NAP approximation in Section 5.4 but we could not provide a simple means to decide how to choose a representative subset of data for a group. For efficiency, we see most potential in parallelizing computation steps and in finding more efficient ways in obtaining (potentially pre-computed) activations for examples that belong to a group of interest. This efficiency improvement is particularly important for future work that applies our technique to large models or more complex data in terms of dimensionality or number of groups. More efficient computation of NAPs is also useful for making the training process visualization more feasible for more difficult applications.

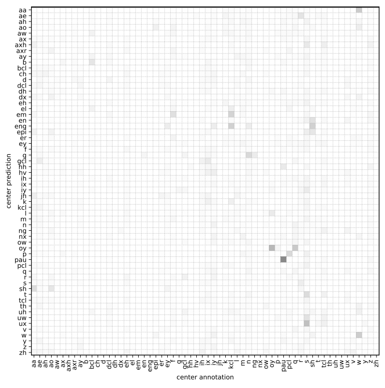
Methodological Improvements Finally, to extend our methodology, we see the highest potential in incorporating information between layers, that are, the weights. At the moment, we only analyze and visualize activity for each layer independently. However, a major part of the information flow and functionality of a DNN is controlled by the weights that connect the layers. Future research is likely to provide substantially more insight into the model when considering the connections between neurons and feature maps, too. For example, activations in a layer could be adjusted based on their influence on the next layer. This way, it becomes better distinguishable whether a highly active neuron has a positive or negative impact on a connected neuron in the next layer. This is, however, a challenging task as there typically are pairwise connections between any pair of neurons (or feature maps) in subsequent layers. Moreover, the visualization becomes difficult because it adds layer depth as a third dimension such that three-dimensional visualizations on screen or in virtual reality might become more suitable.

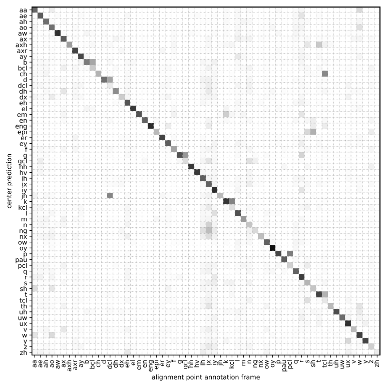
A
Supplemental Figures

A.1 Alignment Evaluation



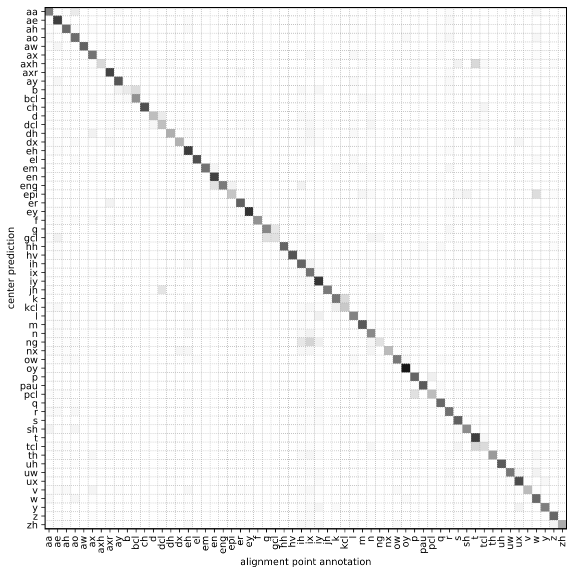
(a) after alignment



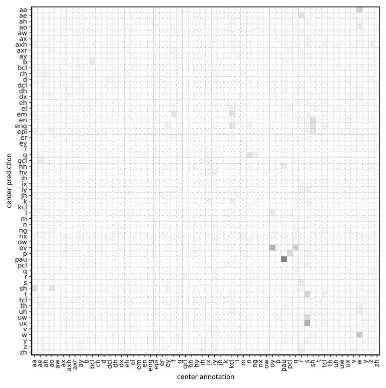


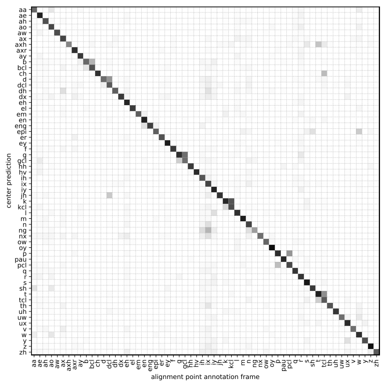
(b) before alignment (c) within alignment frame

Figure A.1: Alignment evaluation overview for model W2P. For each predicted phoneme on the y-axis the relative frequency of the corresponding phoneme annotation on the x-axis. Color scale [0,1] from white to black.



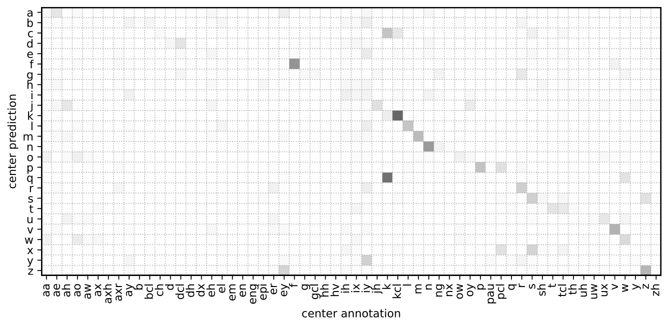
(a) after alignment



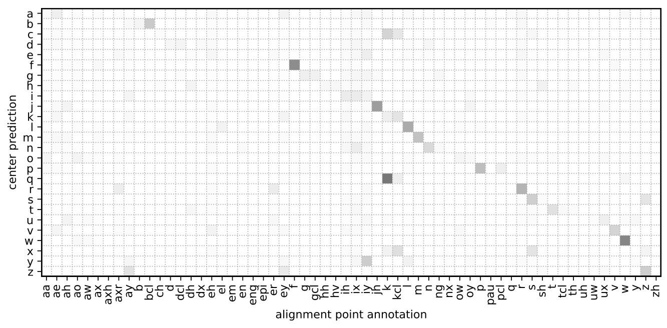


(b) before alignment (c) within alignment frame

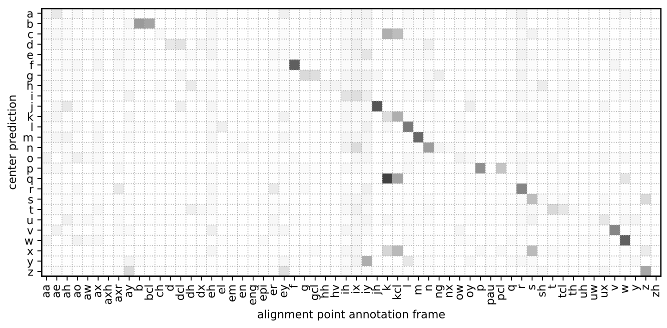
Figure A.2: Alignment evaluation overview for model W2P_shallow. For each predicted phoneme on the y-axis the relative frequency of the corresponding phoneme annotation on the x-axis. Color scale [0,1] from white to black.



(a) before alignment



(b) after alignment



(c) within alignment frame

Figure A.3: Alignment evaluation overview for model W2L. For each predicted letter on the y-axis the relative frequency of the corresponding phoneme annotation on the x-axis. Color scale [0,1] from white to black.

A.2 Approximated NAPs

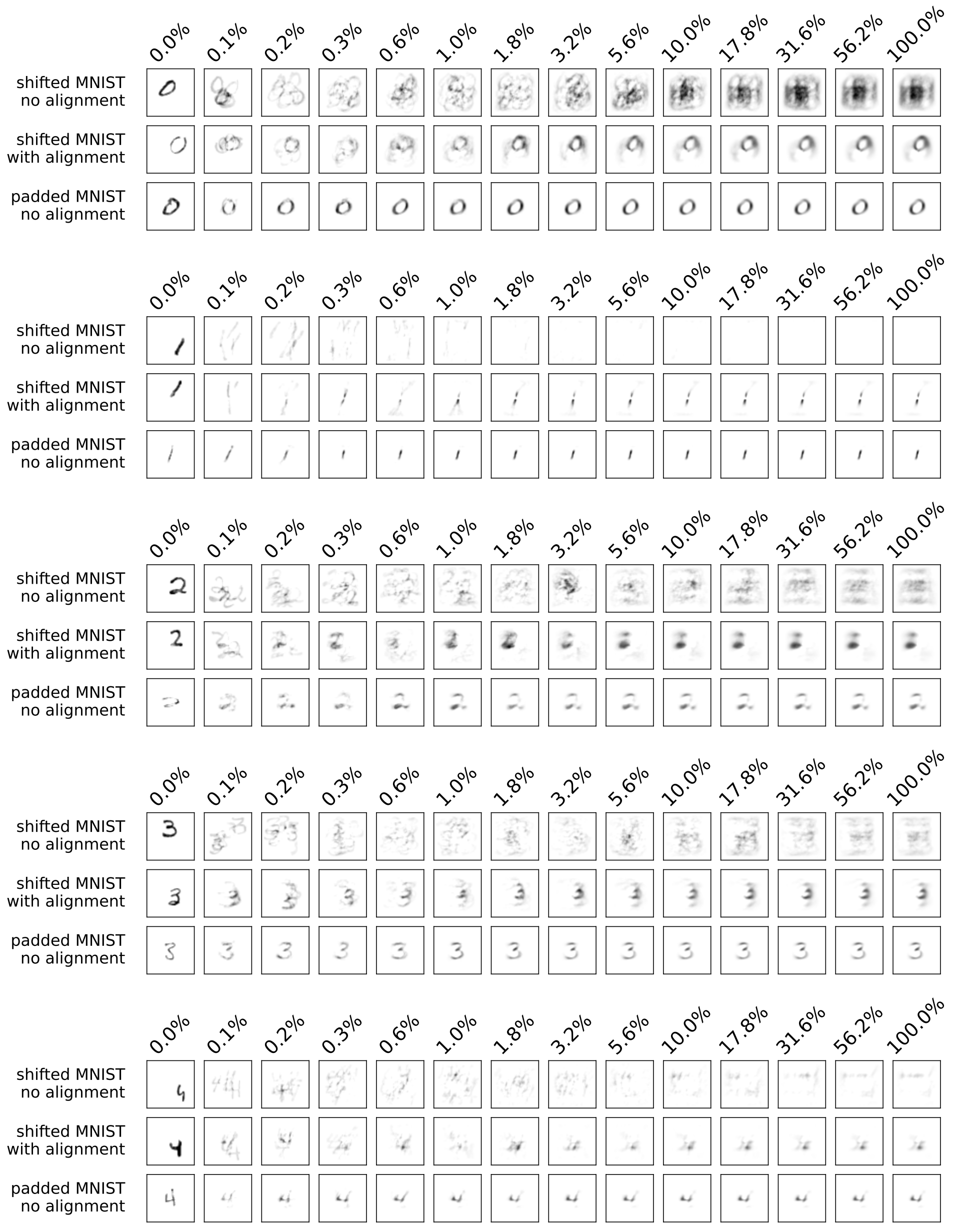


Figure A.4: Approximated input layer NAPs for classes 0–4.

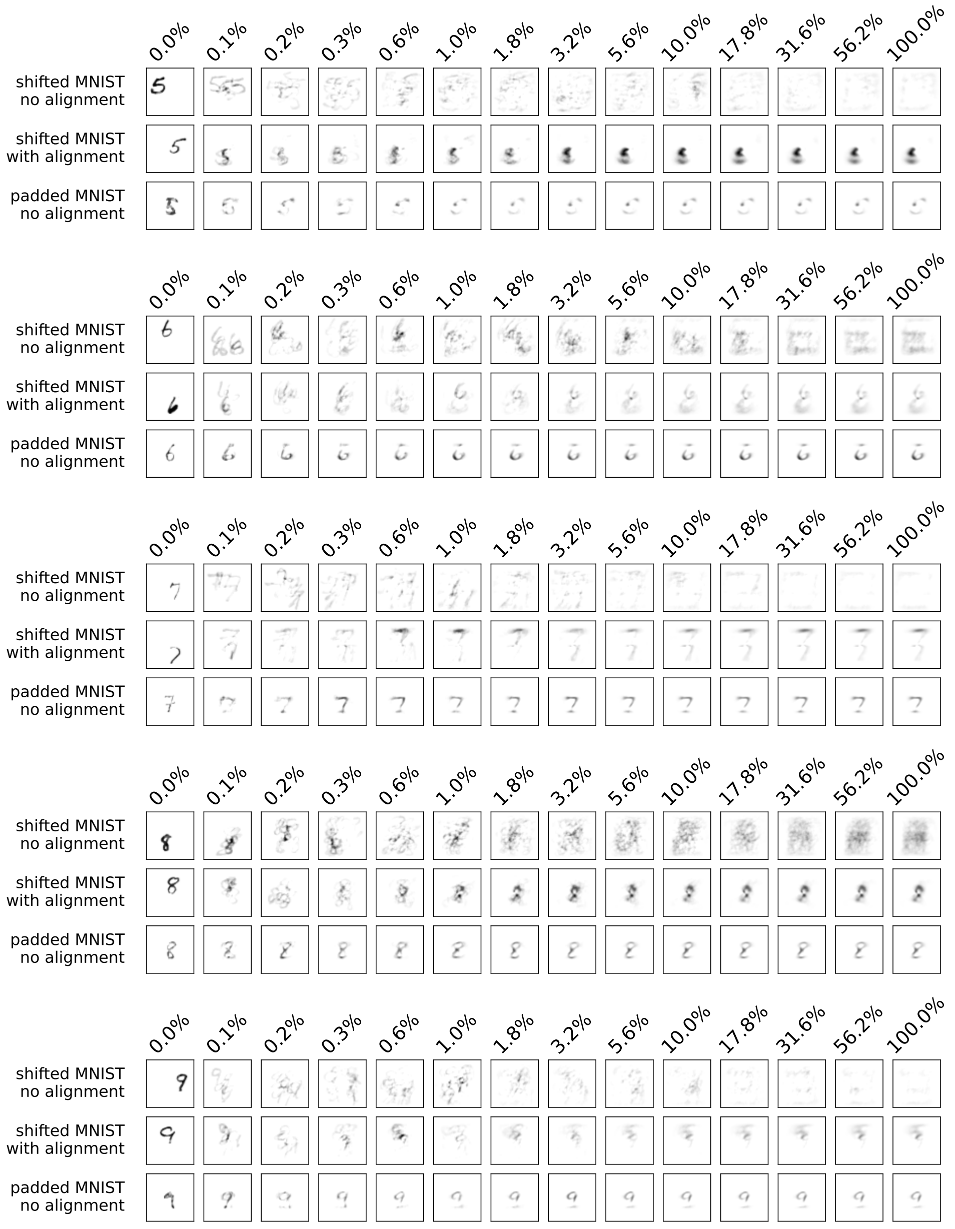


Figure A.5: Approximated input layer NAPs for classes 5–9.

A.3 NAP Approximation Errors

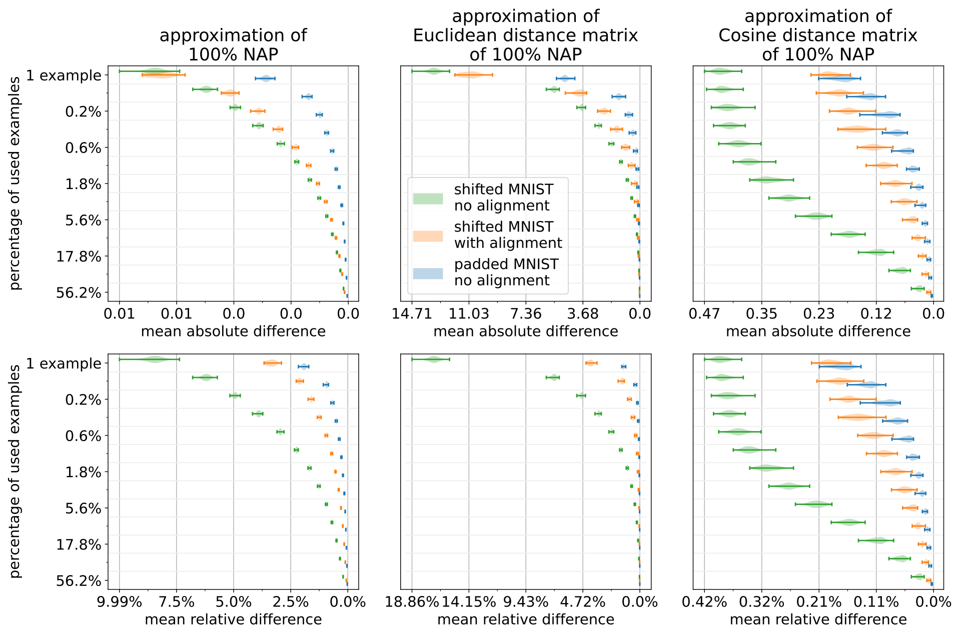


Figure A.6: NAP approximation errors in convolutional layer 1.

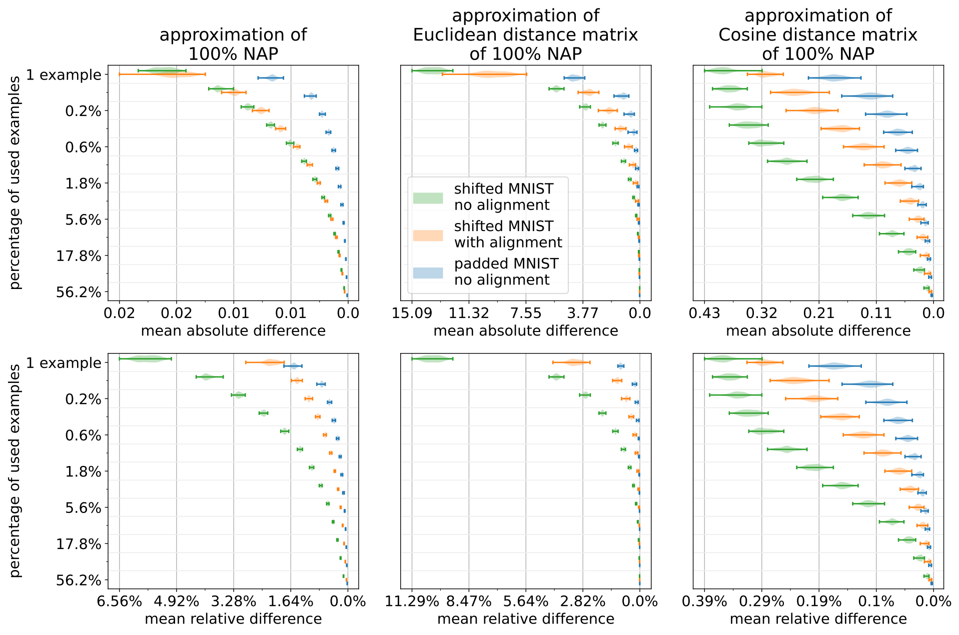


Figure A.7: NAP approximation errors in convolutional layer 2.

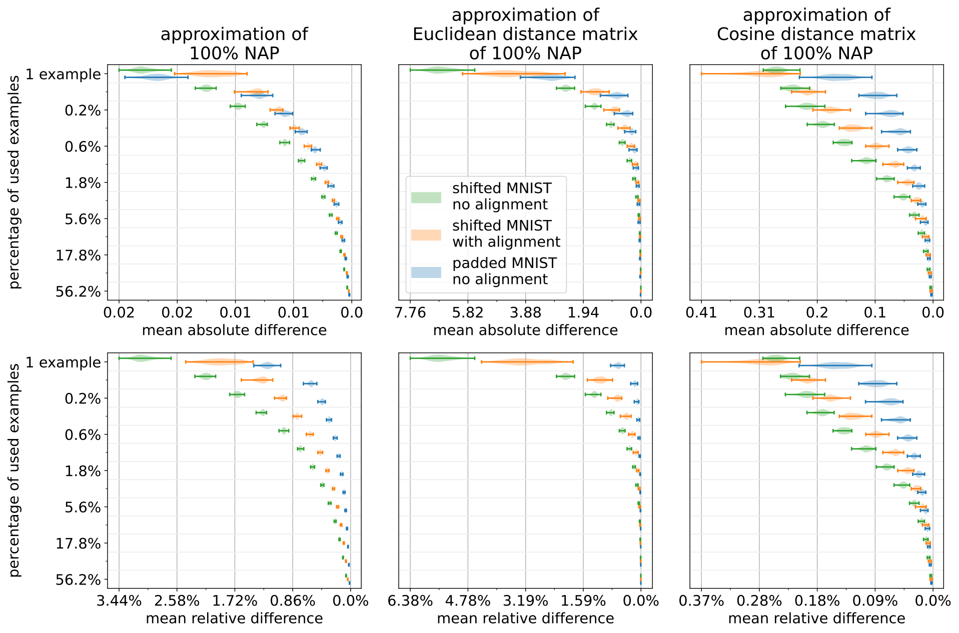


Figure A.8: NAP approximation errors in convolutional layer 3.

A.4 Input Layer NAPs

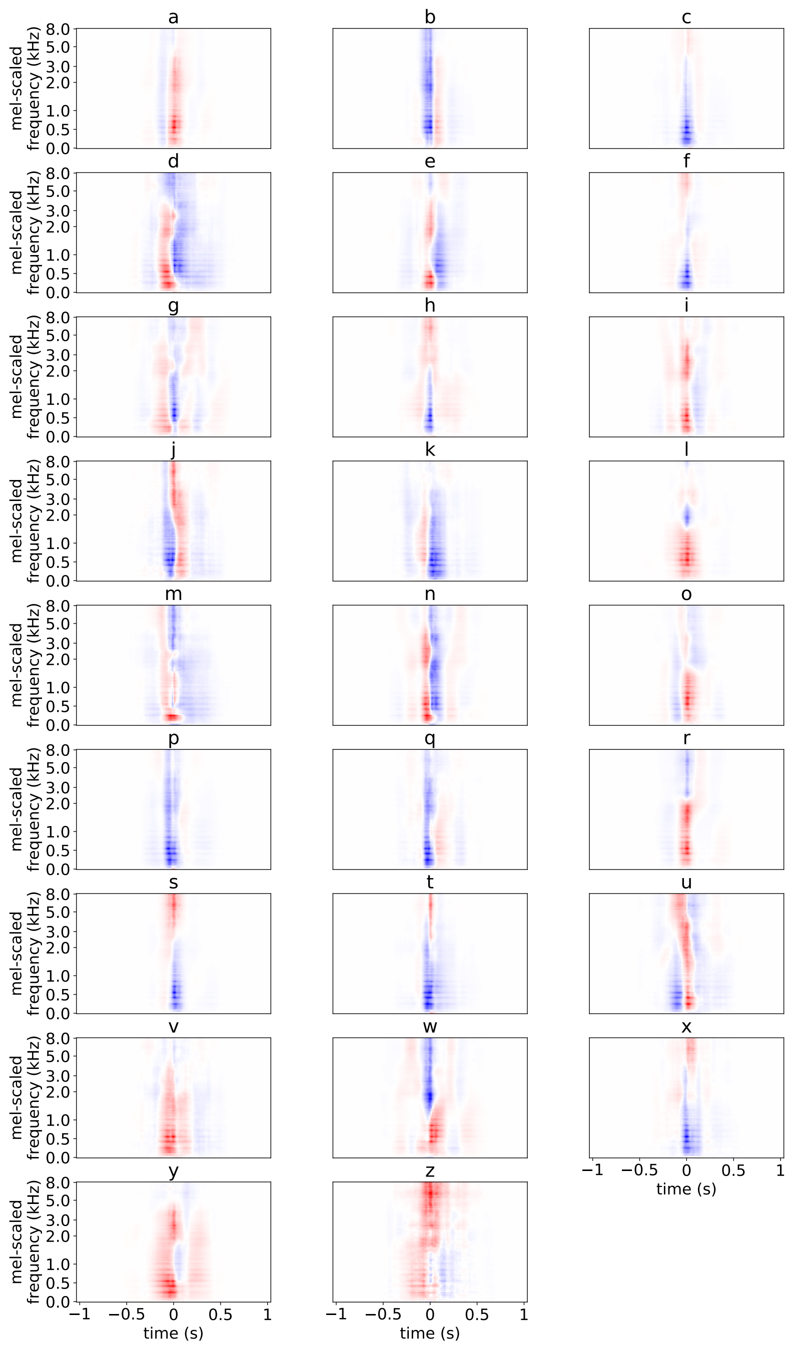


Figure A.9: Input layer SNAPs in model W2L for all letters.

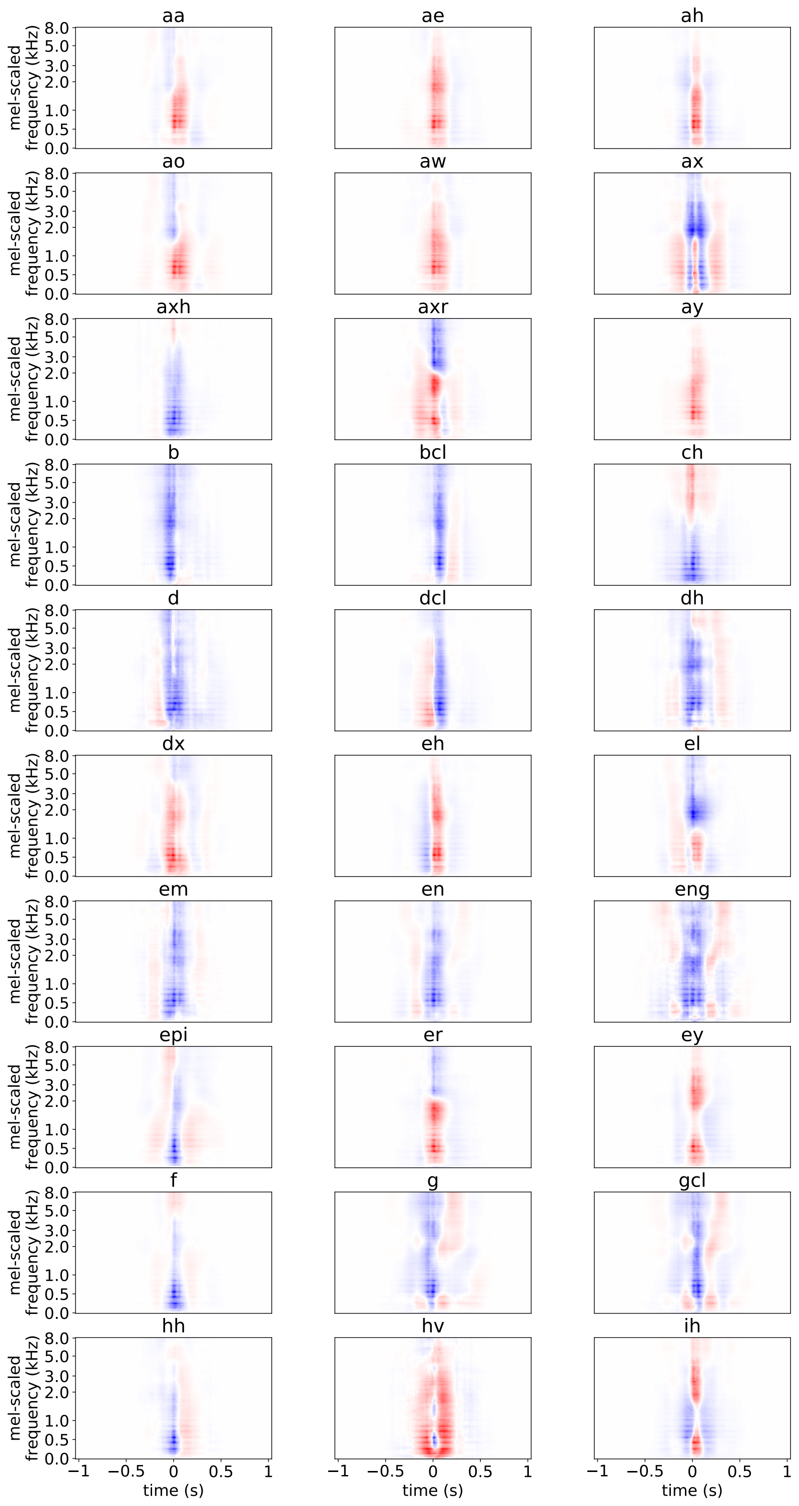


Figure A.10: Input layer SNAPs in model W2L for phonemes `aa`–`ih`.

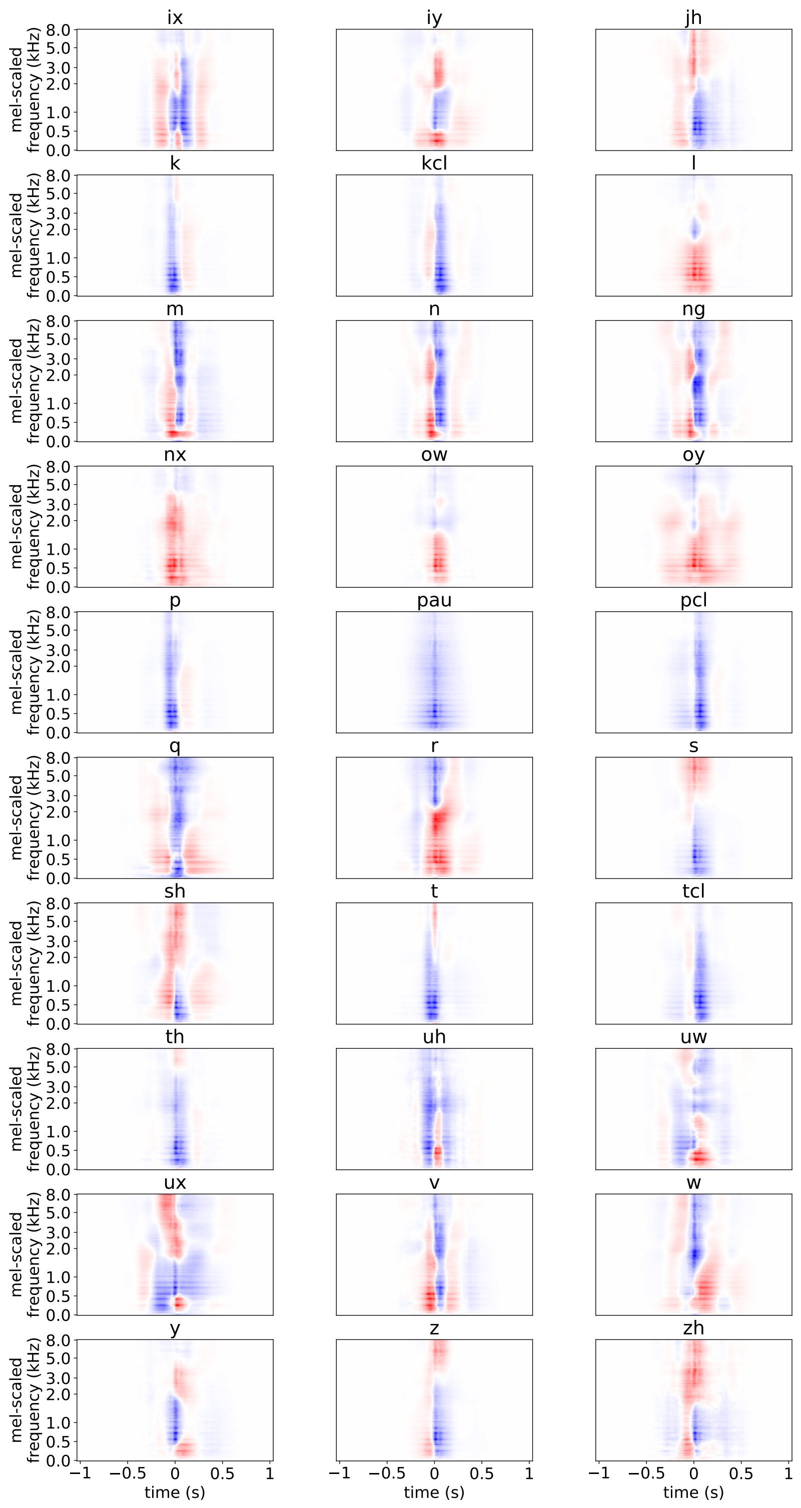


Figure A.11: Input layer SNAPs in model W2L for phonemes `ix`–`zh`.

A.5 VGG16 Topographic Maps in Different Layers

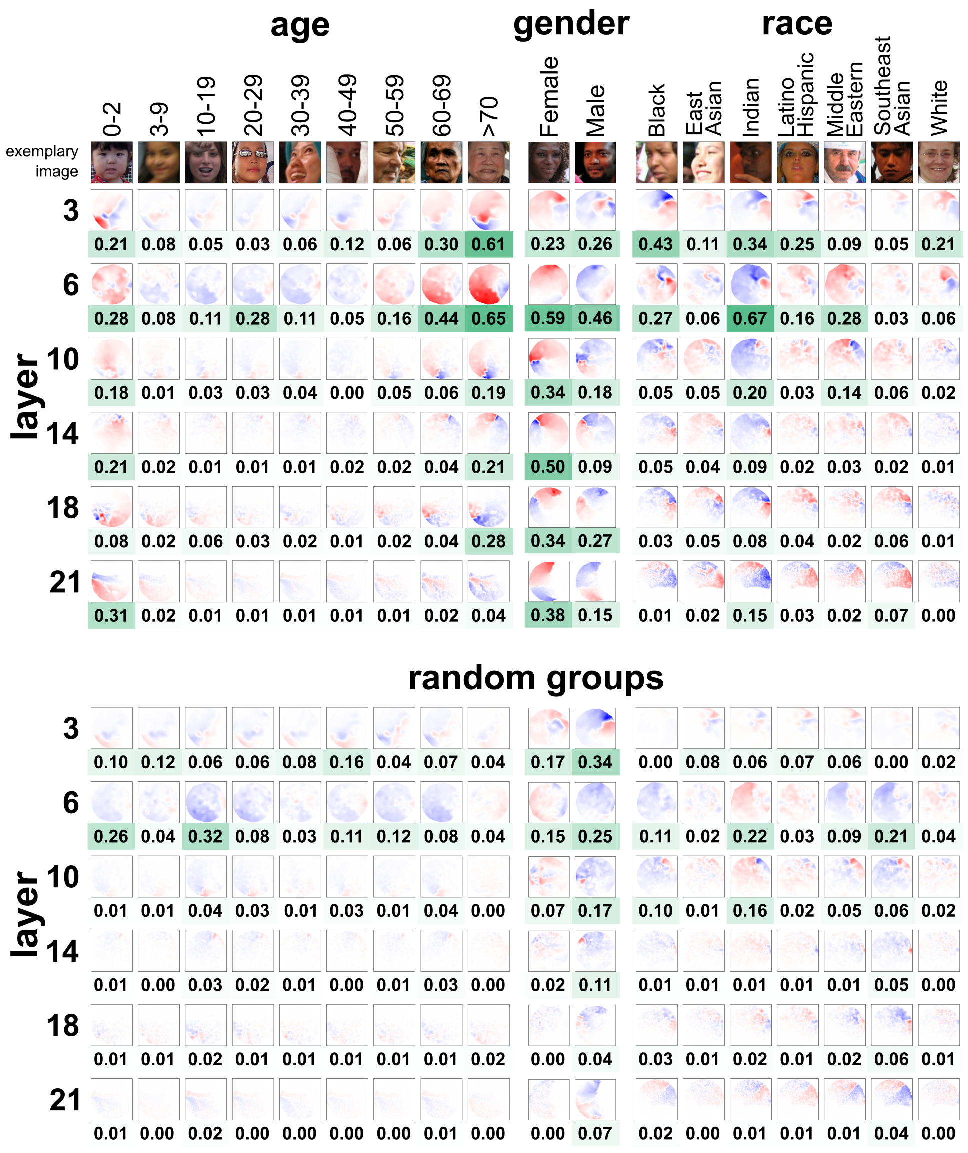
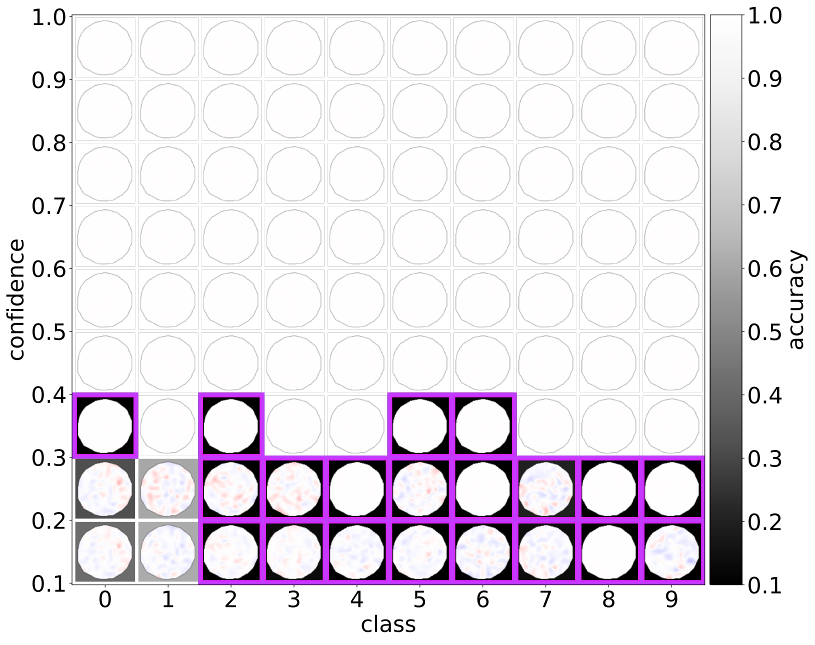


Figure A.12: Topographic maps of VGG16 activations in all maxpooling layers (layers 3, 6, 10, 14, 18) and the last fully-connected layer (layer 21) for the FairFace categories, as well as random groups for comparison. Size-weighted convexity values are shown for each topographic map.

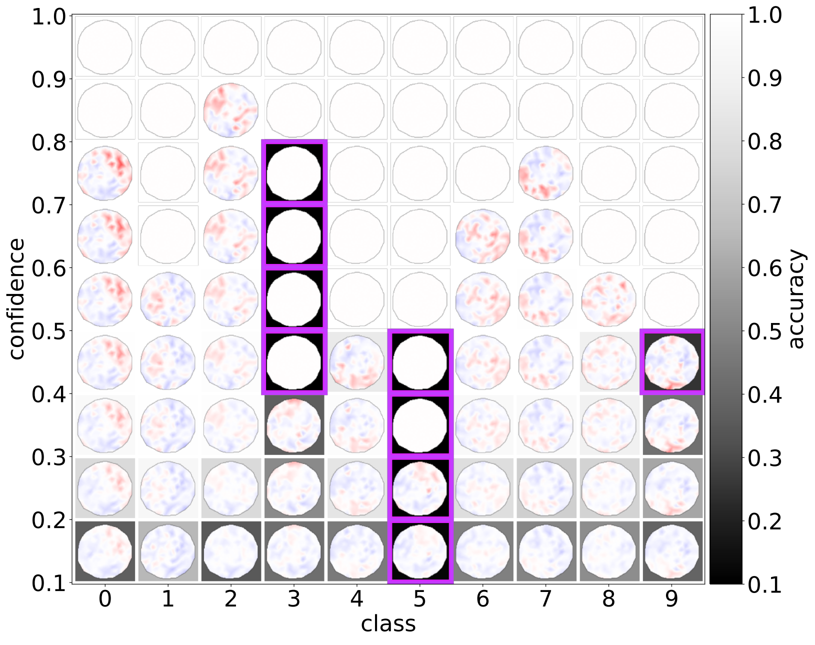
A.6 Confidence and Activation over Training Time

This Appendix section shows topographic maps for confidence- and class-separated groups at different steps in training. Background heat maps show accuracy and a purple outline indicate overconfidence. Positions with all-white topographic maps are empty groups or groups without correct predictions.

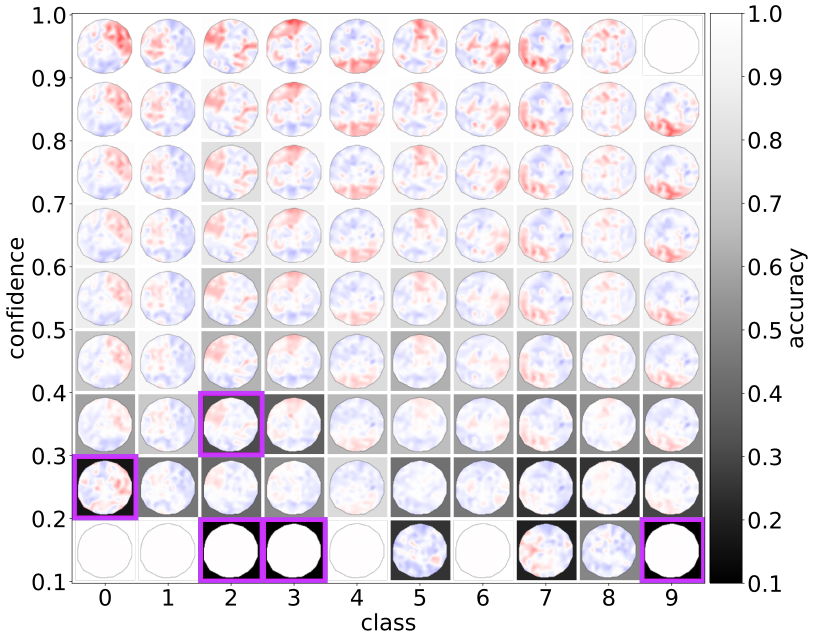
Step 1



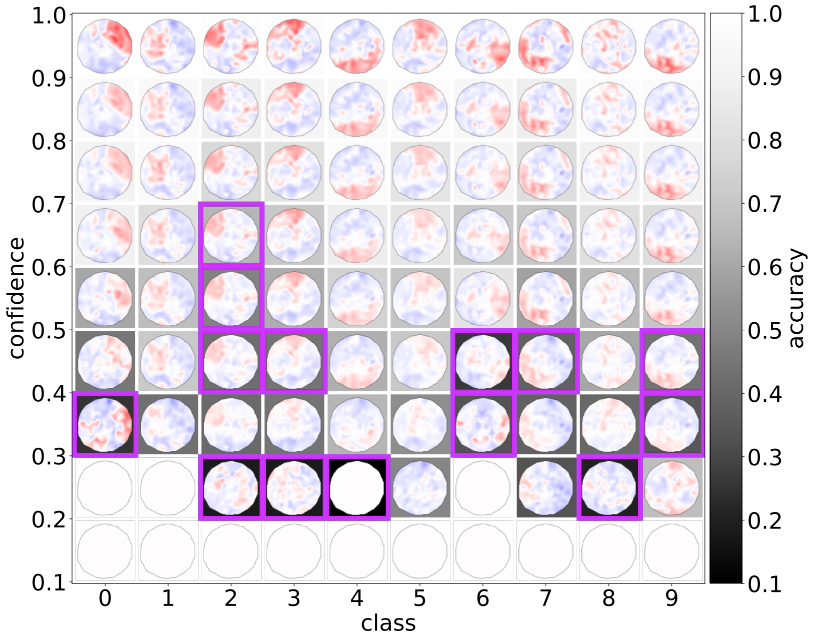
Step 20



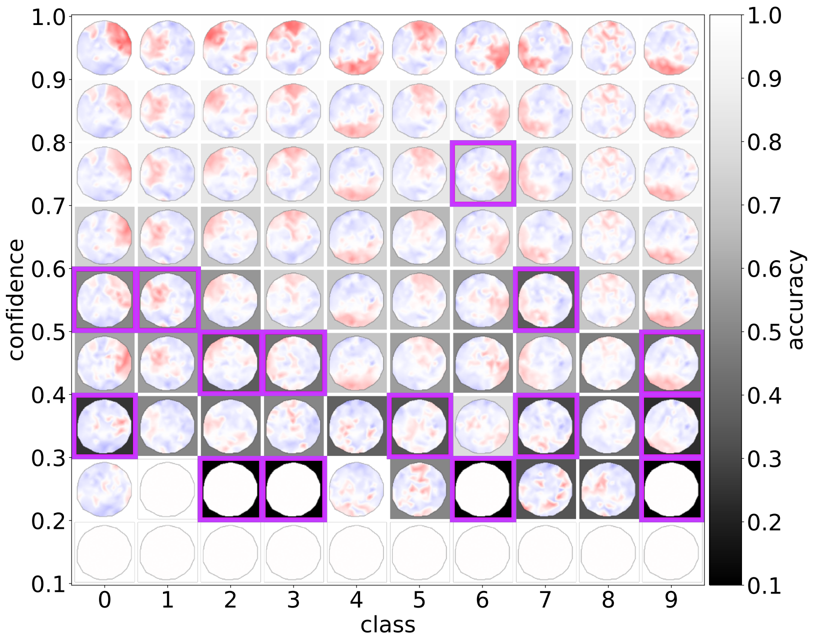
Step 120



Step 500



Step 1875


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
confidence
0
1
2
3
4
5
6
7
8
9
class
accuracy

B
Supplemental Results

B.1 Extended Quantitative Topographic Map Evaluation

In this section, we report the quantitative visual quality evaluation of topographic activation maps from Section 6.3.3 for all investigated measures. We focus on discussing all but the size-weighted convexity which is already discussed in the main part but include it in the figures for comparison.

MLP

For the PSO-based layouting techniques, we quantify visual quality of topographic maps with different measures. The results for the first fully-connected layer of the MLP model trained on MNIST are shown in Figure B.1.

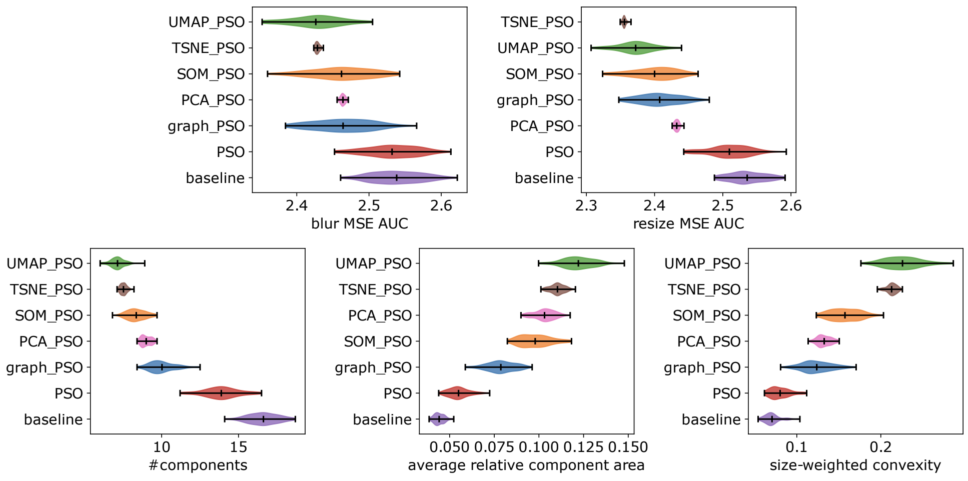


Figure B.1: Quantification of the topographic map quality for MNIST MLP.

The lower end of the quality for all measures is shared by random_PSO and the PSO with random initialization. With respect to the blur and resize MSE there is no significant quality difference between these two layouting techniques. Regarding the number and size of the components, PSO performs slightly better than the random baseline but is still of significantly worse quality than all PSO with non-random initialization. This supports our finding from the method pre-selection.

SOM_PSO, graph_PSO, and PCA_PSO are in the medium quality range for all metrics, while graph_PSO shows the lowest quality among the three methods. The quality differences of SOM_PSO and PCA_PSO are small and different quality metrics identify either one or the other as the better technique. Observing both the number of components and their size, SOM_PSO results in fewer and smaller components than PCA_PSO. Fewer components are only advantageous if they are also larger. In addition, PCA_PSO also has a lower variation than SOM_PSO. Therefore, we consider PCA_PSO as a higher-quality layouting method than SOM_PSO.

UMAP_PSO and TSNE_PSO obtain the highest quality results for the MLP according to all metrics. TSNE_PSO shows smaller MSE when blurring or resizing the topographic map than UMAP_PSO, however not to a significant amount. Regarding the number and size of the components UMAP_PSO has a clearly higher quality as it shows fewer components of a larger size than TSNE_PSO. Note that topographic map quality for UMAP_PSO shows a high variation, such that individual layouts using this technique might not perform better than TSNE_PSO.

In conclusion, considering the higher quality regarding the component-based metrics, we find UMAP_PSO the layouting metric of highest quality.

CNN

First we perform the same quality comparison of using different layouting method as for the MLP example. In CNNs, the quality of the topographic maps can further be influenced by how the feature maps are aggregated to obtain color values, therefore, we also use different aggregation approaches. For some methods, random aggregation leads to extreme values such that differences between mean, max and center aggregation are not visible from the plots anymore. Therefore, we omitted random aggregation for the comparison of all layouting methods. Figures B.2 and B.3 show the results of the quality comparison as an overview of average quality rank across the aggregation methods (Figure B.2) and as the detailed quality metric distributions (Figure B.3).

On average across all aggregations, UMAP_PSO shows the best rank according to all quality metrics. This indicates that UMAP_PSO is most suitable as layouting technique for both MLPs and CNNs. Different to our findings for MLPs, TSNE_PSO is not as high-quality as UMAP_PSO anymore and even is lower-quality than SOM_PSO or PCA_PSO in many cases. PSO with random initialization leads to low-quality topographic maps as in the MLP. However, for CNNs, graph_PSO now is not producing significantly better results than PSO with random initialization.

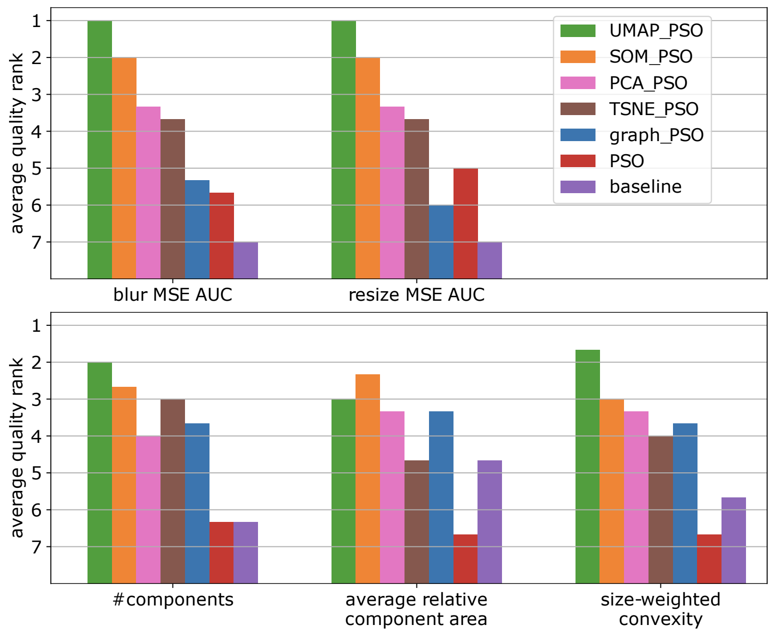


Figure B.2: Overview of quality values for padded MNIST topographic maps for different layouting techniques as average rank over the aggregation techniques using the average quality values over the 100 repetitions.

In the detailed view of Figure B.3, we focus on cases where individual observations deviate from the trend observed in the overview of Figure B.2. Firstly the rank does not always reflect whether the qualities of layouting techniques differ significantly. For example, while UMAP_PSO has a higher quality than PCA_PSO according to the rank, their quality is not significantly different according to the component count and size (using mean and center aggregation). Secondly, it is prominent that there is a high similarity between the results for mean and center aggregation but not for max aggregation. Comparing mean and center aggregation, we observe a similar quality order of the layouting techniques, only the range of the quality values differs. In contrast, using the maximum for aggregating the feature maps only leads to similar order according to the blur and resize MSE, but to highly different quality order in the component metrics. For example, using max aggregation, the random layout appears to perform well as it shows large and few components. However, this indicates a weakness of this aggregation technique. Due to only aggregating to the highest activation value in a feature map, all obtained color values are in the red (positive) range. Consequently, it is very prone to only producing a large

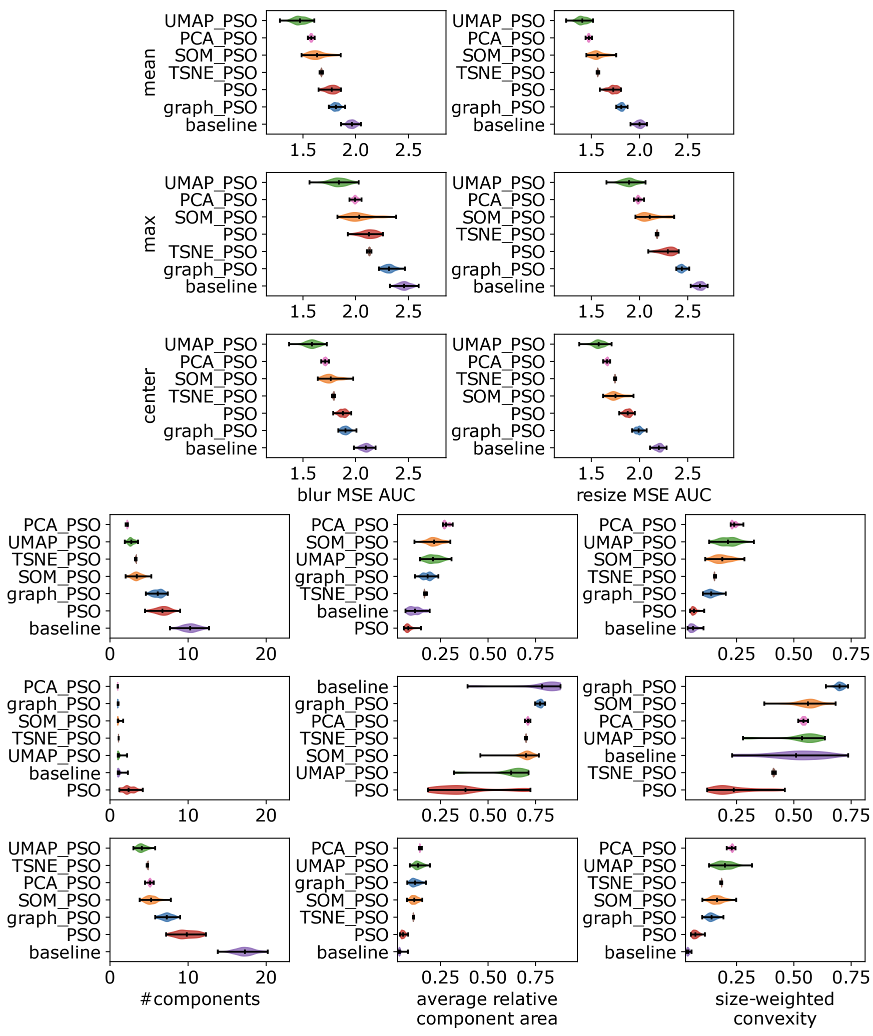


Figure B.3: Detailed quality metric distributions for padded MNIST topographic maps for different layouting techniques over all repetitions for all aggregations.

red component. While this leads to good component-based quality values, the actual topographic maps are useless as there are no distinguishable regions and differences between groups are less pronounced. This is a strong indication that a global max pooling of feature maps is not suitable for obtaining useful aggregated activation values. Excluding the max aggregation UMAP_PSO produces the highest-quality topographic maps according to all metrics and both mean and center aggregation, with only insignificantly better quality values for other techniques.

Because UMAP_PSO is shown to produce the highest-quality topographic maps according to the different metrics, we continue to use it in the following comparisons and experiments.

Alignment and Aggregation

Creating topographic activation maps for CNNs differs from MLPs mainly in in that feature map NAPs values need to be aggregated to obtain colors. How well this works might depend on whether the inputs or feature maps are aligned and which aggregation function is used. Here, we first investigate whether there are differences using aligned or unaligned data. Second, we test the suitability of different aggregation methods in dependence of the alignment of the data.

MNIST Variations and Alignment In this section, we investigate how the quality is affected by whether the used data is pre-aligned, not aligned or aligned using our gradient-based technique. The topographic map qualities for the three alignment types, using UMAP_PSO as layouting technique, are shown in Figure B.4.

In general, there is no clear trend of one alignment variation producing generally better topographic maps than others. For MNIST with our gradient-based alignment, we observe better topographic maps than with pre-aligned data according to component-based metrics and center and max aggregation. In other cases, using qualities of topographic maps using our alignment technique are either similar or slightly worse than with pre-aligned data. Surprisingly, only for mean aggregation and according to MSE-based quality measure, the shifted MNIST version without alignment performs worse than than the pre-aligned data. In all other cases it either produces similar or better quality topographic maps than with padded MNIST. From these results, we conclude that the topographic map quality does not strongly depend on whether the data are aligned or not. The alignment rather affects the representativeness of the NAPs but the topographic maps are still of good quality. Consequently, of course, the topographic maps based on NAPs of not aligned data can be less representative, as well.

Aggregation Methods Here, we compare whether the choice of aggregation technique has an influence on the topographic map quality, in dependence of how the data are aligned. We include the random aggregation technique

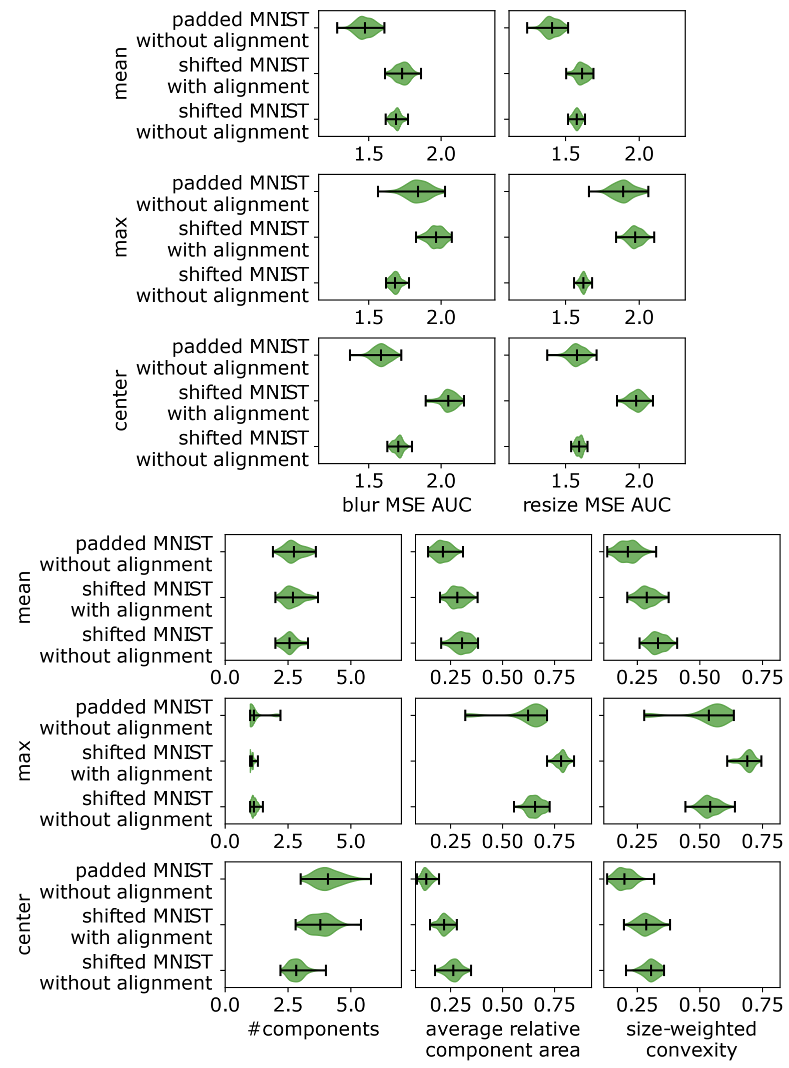


Figure B.4: Topographic map qualities using UMAP_PSO comparing MNIST alignment variations.

as it does not produce very strong outlier values for UMAP_PSO. The quality value distributions are shown in Figure B.5.

First, we investigate the shifted MNIST without alignment (bottom row) as it showed surprisingly high quality in the previous section. We observe that there is little influence of which aggregation technique is used for this data. According to the MSE-based quality measures, either aggregation leads so similarly well results, including the random aggregation. Regarding the component-based measures, we can observe some differences.

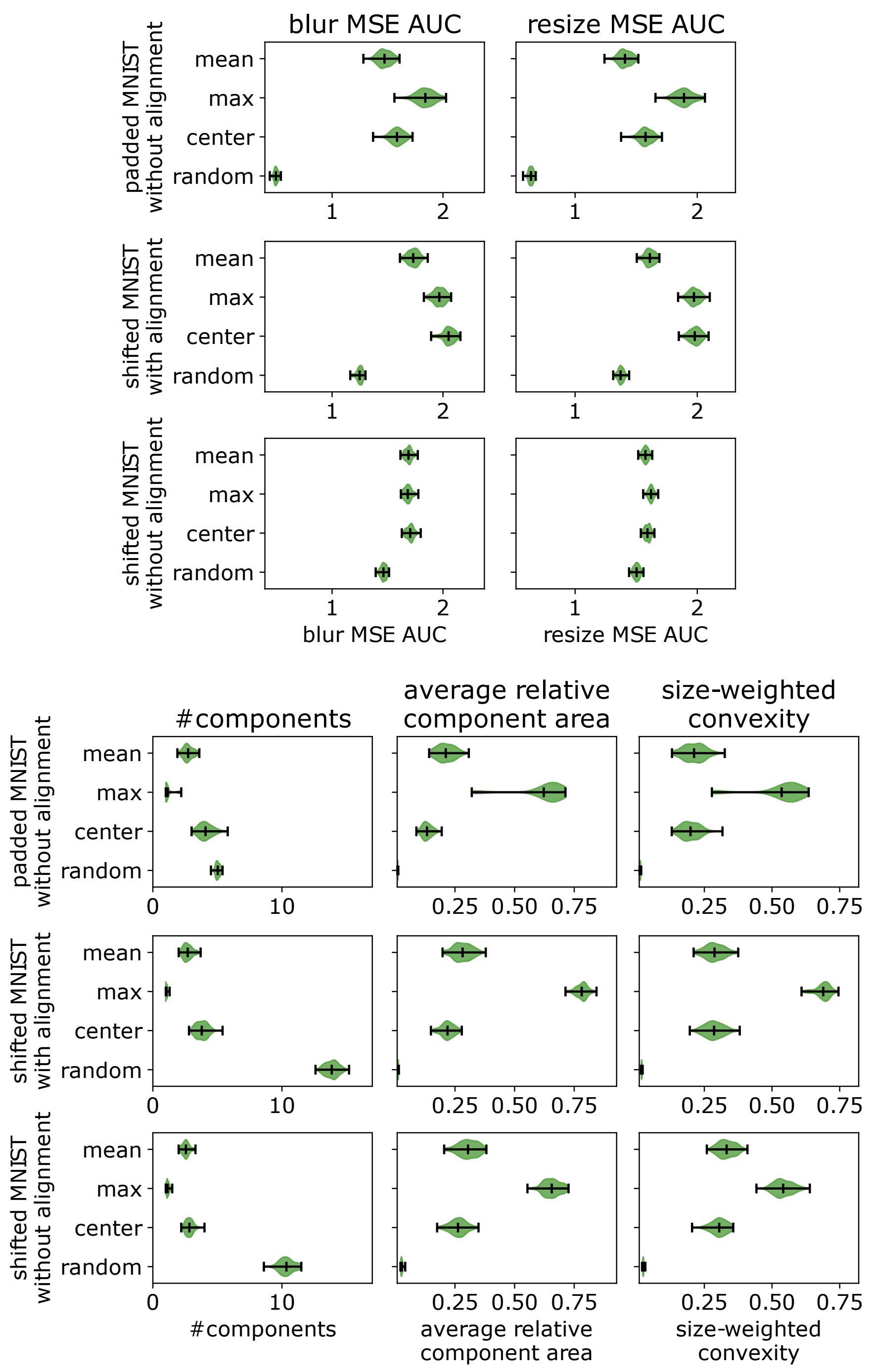


Figure B.5: Topographic map qualities using UMAP_PSO comparing aggregation methods.

While the number of components is similar for all aggregations except random, max aggregation shows significantly larger components. As discussed above, this is due to max aggregation implies that there are only few large components in the positive NAP values. The small difference between the aggregation techniques for this data supports that the underlying NAP values are not informative.

For the aligned data (pre-aligned and with our gradiend-based alignment), there are clearer differences between the aggregation techniques. Both data sets show low MSE-based quality for max aggregation, but high component-based quality. This is again the effect of the large positive-value components that max aggregation produced by design. However there is no clear variation between the groups which makes the topographic maps less useful. Therefore, we consider max aggregation to be not suitable despite the good quality values. Moreover, for the two aligned data sets, there is also a clear difference between the intelligent aggregation techniques and the random one. This indicates, in contrast to the not aligned data, that the NAP values are more informative for both aligned data sets.

We expected that center aggregation is well-suitable for our alignment technique because it aligns the most important feature map position to the center. If this was the case, center aggregation would show a relatively better performance for shifted MNIST with alignment compared to padded MNIST which is pre-aligned. Padded MNIST shows slightly better topographic maps for mean than for center aggregation but the differences are insignificant for MSE-based quality measures and only small for the component-based qualities. For shifted MNIST that is aligned with our technique, center aggregation leads to clearly lower-quality topographic maps than center aggregation for the MSE-based qualities, while the component-based metrics indicate more similar quality. In summary, there is a difference in using mean or center aggregation when using pre-aligned data or applying our alignment technique. However, from the results, it is not evident that using center aggregation brings a significant improvement of quality when using our gradient-based alignment.

Based on the results, we conclude that using the mean value is the most suitable and generally applicable aggregation technique as it performs similarly well regardless of how and whether the data are aligned.

Layer Comparison

Using the most promising layouting technique (UMAP_PSO) and the most suitable aggregation function (mean), we further compare visual quality of topographic activation maps in different layers of the network. Results are shown in Figure B.6.

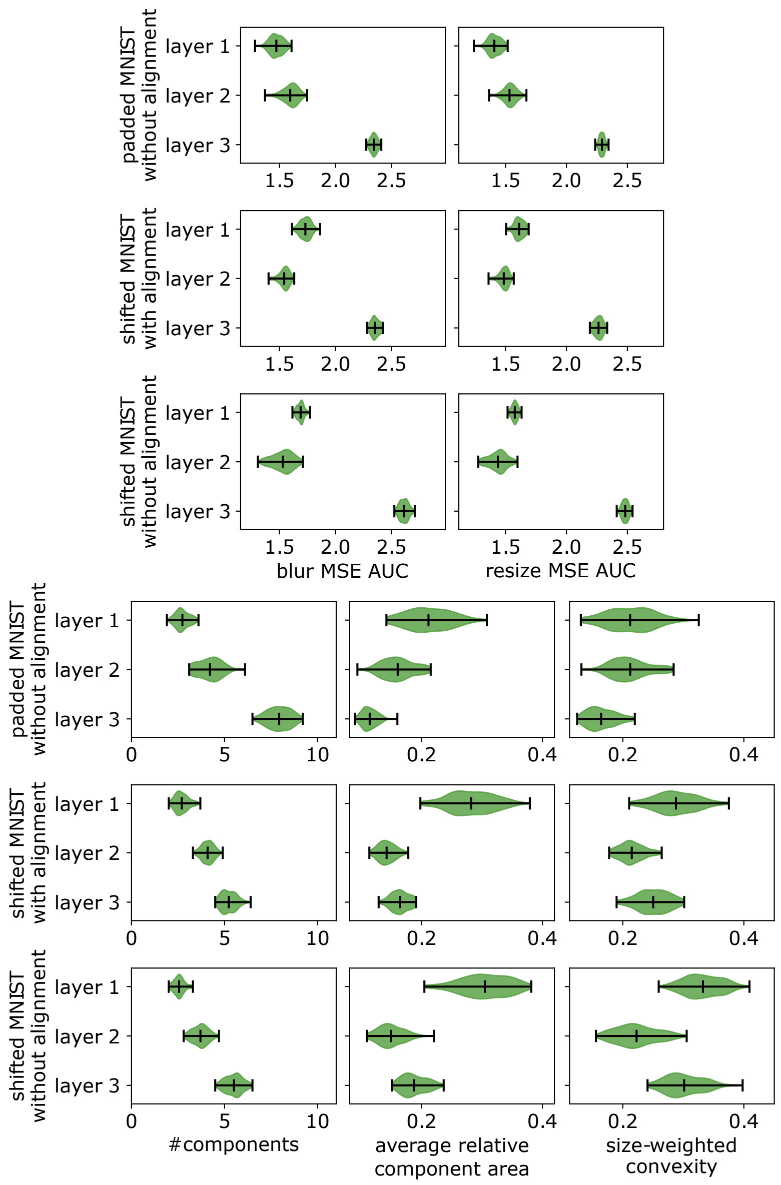


Figure B.6: Topographic map qualities using UMAP_PSO and mean aggregation comparing layers.

We observe that visual quality is generally lower in deeper layers. Especially, the third layer quality drops for all measures except the size-weighted convexity. For both shifted MNIST, the second layer appears visually worse than the first layer but better for padded MNIST. In the particular model, we used layers with the same number of feature maps but they decrease in size due to the used stride of 2. Likely, aggregating the smaller feature maps in deeper layers yield more class-specific information about the activations and therefore lead to more diverse topographic maps. This higher diversity of the regions might lead to more separated regions which our measures consider lower visual quality.

C List of Abbreviations

AE	Autoencoder
ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
AUC	area under the curve
BM	Boltzmann Machine
CAM	Class Activation Mapping
CCA	Canonical Correlation Analysis
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
DL	Deep Learning
DNN	Deep Neural Network
EEG	Electroencephalography
ERP	Event-Related Potential
GAN	Generative Adversarial Network
Grad-CAM	Gradient-weighted Class Activation Mapping
GradNAP	Gradient-Adjusted Neuron Activation Profile
GRU	Gated Recurrent Unit
IPA	International Phonetic Alphabet

LIME	Local Interpretable Model-agnostic Explanations
LRP	layer-wise relevance propagation
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NAP	Neuron Activation Profile
PCA	Principal Component Analysis
PSO	Particle Swarm Optimization
RA	Research Aim
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrrent Neural Network
SHAP	SHapley Additive exPlanation
SNAP	Saliency-Adjusted Neuron Activation Profile
SOM	Self-Organizing Map
STFT	Short-time Fourier transform
SVD	Singular Value Decomposition
tSNE	t-Distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection
W2L	Wav2Letter
XAI	explainable Artificial Intelligence

D
List of Figures

E
List of Tables

F

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL `https://www.tensorflow.org/`.

[2] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[3] D. M. Abdullah and N. S. Ahmed. A review of most recent lung cancer detection techniques using machine learning. *International Journal of Science and Business*, 5(3):159–173, 2021.

[4] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, pages 9525–9536, 2018.

[5] J. Ahn and A. Oh. Mitigating language-dependent ethnic bias in BERT. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 533–549, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics.

[6] G. Alain and Y. Bengio. Understanding intermediate layers using linear classifier probes. In *International Conference on Learning Representations (ICLR), Workshop Track Proceedings*, 2017.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[8] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. In *2017 International Conference on Computing Networking and Informatics (ICCNI)*, pages 1–9. IEEE, 2017.

[9] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[10] A. M. Badshah, J. Ahmad, N. Rahim, and S. W. Baik. Speech emotion recognition from spectrograms with deep convolutional neural network. In *IEEE International Conference on Platform Technology and Service (PlatCon)*, pages 1–5, 2017.

[11] S. Becker, M. Ackermann, S. Lapuschkin, K.-R. Müller, and W. Samek. Interpreting and explaining deep neural networks for classification of audio signals. *arXiv preprint arXiv:1807.03418*, 2018.

[12] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[13] T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai. Man is to computer programmer as woman is to homemaker? Debiasing Word Embeddings. In *Advances in Neural Information Processing Systems*, volume 29, pages 4349–4357, 2016.

[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

[15] J. Buolamwini and T. Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, volume 81, pages 77–91. PMLR, 23–24 Feb 2018.

[16] A. Byerly, T. Kalganova, and R. Ott. The current state of the art in deep learning for image classification: A review. In *Science and Information Conference*, pages 88–105. Springer, 2022.

[17] S. Carter, Z. Armstrong, L. Schubert, I. Johnson, and C. Olah. Activation atlas. *Distill*, 4(3):e15, 2019.

[18] S. Chatterjee, F. Saad, C. Sarasaen, S. Ghosh, V. Krug, R. Khatun, R. Mishra, N. Desai, P. Radeva, G. Rose, S. Stober, O. Speck, and A. Nürnberger. Exploration of Interpretability Techniques for Deep COVID-19 Classification using Chest X-ray Images. *arXiv preprint arXiv:2006.02570*, 2020.

[19] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian. Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 839–847. IEEE, 2018.

[20] G. Chen, X. Na, Y. Wang, Z. Yan, J. Zhang, S. Ma, and Y. Wang. Data Augmentation For Children's Speech Recognition – The "Ethiopian" System For The SLT 2021 Children Speech Recognition Challenge. *arXiv preprint arXiv:2011.04547*, 2020.

[21] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179.

[22] F. Chollet. *Deep learning with Python.* Simon and Schuster, 2021.

[23] CMUdict. Carnegie mellon pronouncing dictionary, 2014.

[24] R. Collobert, C. Puhrsch, and G. Synnaeve. Wav2Letter: an End-to-End ConvNet-based Speech Recognition System. *arXiv preprint arXiv:1609.03193*, 2016.

[25] N. Cummins, S. Amiriparian, G. Hagerer, A. Batliner, S. Steidl, and B. W. Schuller. An image-based deep spectrum feature representation for the recognition of emotional speech. In *ACM International Conference on Multimedia*, pages 478–484, 2017.

[26] T. Czimmermann, G. Ciuti, M. Milazzo, M. Chiurazzi, S. Roccella, C. M. Oddo, and P. Dario. Visual-based defect detection and classification approaches for industrial applications — a survey. *MDPI Sensors*, 20(5):1459, 2020.

[27] S. Dargan, M. Kumar, M. R. Ayyagari, and G. Kumar. A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering*, pages 1–22, 2019.

[28] K. Deng, S. Cao, Y. Zhang, L. Ma, G. Cheng, J. Xu, and P. Zhang. Improving CTC-based speech recognition via knowledge transferring from pre-trained language models. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8517–8521. IEEE, 2022.

[29] M. Deshmukh and U. Bhosle. A survey of image registration. *International Journal of Image Processing (IJIP)*, 5(3):245, 2011.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2019.

[31] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[32] R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 84–88 vol.1, 2000.

[33] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[34] D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.

[35] L. Ericsson, H. Gouk, C. C. Loy, and T. M. Hospedales. Self-supervised representation learning: Introduction, advances, and challenges. *IEEE Signal Processing Magazine*, 39(3):42–62, 2022.

[36] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *kdd*, 96(34):226–231, 1996.

[37] R. Fan, W. Chu, P. Chang, and A. Alwan. Transformer for end-to-end automatic speech recognition. *IEEE Transactions on Audio, Speech, nd Language Processing*, page 1, 2023.

[38] J. Fiacco, S. Choudhary, and C. Rose. Deep neural model inspection and comparison via functional neuron pathways. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5754–5764, 2019.

[39] A. Fisher, C. Rudin, and F. Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *J. Mach. Learn. Res.*, 20, 177:1–81, 2019.

[40] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[41] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93:27403, 1993.

[42] J. M. Gauch. Noise removal and contrast enhancement. In *The Colour Image Processing Handbook*, pages 149–162. Springer, 1998.

[43] S. Gollapudi. *Practical Machine Learning.* Packt Publishing Ltd, 2016.

[44] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[45] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[46] J. C. Gower. Generalized procrustes analysis. *Psychometrika*, 40: 33–51, 1975.

[47] A. Graikos, N. Malkin, N. Jojic, and D. Samaras. Diffusion models as plug-and-play priors. In *Advances in Neural Information Processing Systems*, volume 35, pages 14715–14728, 2022.

[48] M. F. A. Hady and F. Schwenker. Semi-supervised learning. *Handbook on Neural Information Processing*, pages 215–239, 2013.

[49] R. M. Hanifa, K. Isa, and S. Mohamad. A review on speaker recognition: Technology and challenges. *Computers & Electrical Engineering*, 90:107005, 2021.

[50] G. Harshvardhan, M. K. Gourisaria, M. Pandey, and S. S. Rautaray. A comprehensive survey and analysis of generative models in machine learning. *Computer Science Review*, 38:100285, 2020.

[51] S. Hart. Shapley value. In *Game theory*, pages 210–216. Springer, 1989.

[52] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[53] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[54] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning. Lecture 6a. Overview of mini-batch gradient descent, 2012.

[55] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. ISSN 0899-7667.

[56] F. Hohman, H. Park, C. Robinson, and D. H. P. Chau. Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1096–1106, 2019.

[57] F. Hohman, H. Park, C. Robinson, and D. H. P. Chau. Summit tool, 2019. URL `https://fredhohman.com/summit/`.

[58] M. Holmstrom, D. Liu, and C. Vo. Machine learning applied to weather forecasting. *Meteorol. Appl*, 10:1–5, 2016.

[59] A. Holzinger, A. Saranti, C. Molnar, P. Biecek, and W. Samek. Explainable AI methods - a brief overview. In *International Workshop on Extending Explainable AI Beyond Deep Models and Classifiers*, pages 13–38. Springer, 2020.

[60] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:498–520, 1933.

[61] M. B. Hoy. Alexa, Siri, Cortana, and more: an introduction to voice assistants. *Medical Reference Services Quarterly*, 37(1):81–88, 2018.

[62] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[63] C. J. Hughes. Single-instruction multiple-data execution. *Synthesis Lectures on Computer Architecture*, 10(1):1–121, 2015.

[64] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[65] P.-T. Jiang, C.-B. Zhang, Q. Hou, M.-M. Cheng, and Y. Wei. Layercam: Exploring hierarchical class activation maps for localization. *IEEE Transactions on Image Processing*, 30:5875–5888, 2021.

[66] I. Jolliffe. *Principal Component Analysis.* Springer Verlag, New York, 2002.

[67] JustGlowing. MiniSom Package, 2021. URL `https://github.com/JustGlowing/minisom`.

[68] K. Karkkainen and J. Joo. Fairface: Face attribute dataset for balanced race, gender, and age for bias measurement and mitigation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1548–1558, 2021.

[69] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of International Conference on Neural Networks (ICCN)*, volume 4, pages 1942–1948 vol.4, 1995.

[70] M. A. Khan and J. Kim. Toward Developing Efficient Conv-AE-Based Intrusion Detection System Using Heterogeneous Dataset. *MDPI Electronics*, 9(11):1771, 2020.

[71] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, et al. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2668–2677, 2018.

[72] S. Kim, A. Gholami, A. Shaw, N. Lee, K. Mangalam, J. Malik, M. W. Mahoney, and K. Keutzer. Squeezeformer: An efficient transformer for automatic speech recognition. In *Advances in Neural Information Processing Systems*, volume 35, pages 9361–9373, 2022.

[73] P.-J. Kindermans, K. T. Schütt, M. Alber, K.-R. Müller, D. Erhan, B. Kim, and S. Dähne. Learning how to explain neural networks: PatternNet and PatternAttribution. In *International Conference on Learning Representations (ICLR)*, 2018.

[74] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[75] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[76] S. Kiranyaz, T. Ince, O. Abdeljaber, O. Avci, and M. Gabbouj. 1-D Convolutional Neural Networks for Signal Processing Applications. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8360–8364, 2019.

[77] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman. 1D convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021.

[78] H. Kirchhoff and A. Lerch. Evaluation of features for audio-to-audio alignment. *Journal of New Music Research*, 40(1):27–41, 2011.

[79] A. Klautau. ARPABET and the TIMIT alphabet. *An archived file. (Accessed Mar. 12, 2020)*, 2001. URL `https://web.archive.org/web/20160603180727/http://www.laps.ufpa.br/aldebaro/papers/ak_arpabet01.pdf`.

[80] D. Kobak and G. C. Linderman. UMAP does not preserve global structure any better than t-SNE when using the same initialization. *bioRxiv*, 2019.

[81] T. Kohonen. *Self-Organizing Feature Maps*, pages 119–157. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988. ISBN 978-3-662-00784-6.

[82] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang. Quartznet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6124–6128, 2020.

[83] A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-10 (Canadian Institute for Advanced Research), 2009. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[84] A. Krug and S. Stober. Adaptation of the event-related potential technique for analyzing artificial neural networks. In *Cognitive Computational Neuroscience (CCN)*, 2017.

[85] A. Krug and S. Stober. Introspection for convolutional automatic speech recognition. In *EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 187–199, 2018.

[86] A. Krug and S. Stober. Visualizing deep neural networks for speech recognition with learned topographic filter maps. *arXiv preprint arXiv:1912.04067*, 2019. Accepted at ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP.

[87] A. Krug and S. Stober. Gradient-Adjusted Neuron Activation Profiles for Comprehensive Introspection of Convolutional Speech Recognition Models. *arXiv preprint arXiv:2002.08125*, 2020.

[88] A. Krug, R. Knaebel, and S. Stober. Neuron activation profiles for interpreting convolutional speech recognition models. In *NeurIPS Workshop IRASL: Interpretability and Robustness for Audio, Speech, and Language*, 2018.

[89] A. Krug, M. Ebrahimzadeh, J. Alemann, J. Johannsmeier, and S. Stober. Analyzing and visualizing deep neural networks for speech recognition with saliency-adjusted neuron activation profiles. *MDPI Electronics*, 10(11):1350, 2021.

[90] V. Krug, C. Olson, and S. Stober. Visualizing bias in activations of deep neural networks as topographic maps. In *Proceedings of the 1st Workshop on Fairness and Bias in AI (AEQUITAS 2023) co-located with 26th European Conference on Artificial Intelligence (ECAI 2023) Kraków, Poland*. CEUR-WS, 2023. URL `http://ceur-ws.org/Vol-3523/`.

[91] V. Krug, C. Olson, and S. Stober. Relation of activity and confidence when training deep neural networks. *(To appear in) Uncertainty meets Explainability, Workshop at ECML-PKDD 2023, Torino, Italy*, 2023.

[92] V. Krug, R. K. Ratul, C. Olson, and S. Stober. Visualizing deep neural networks with topographic activation maps. In *HHAI 2023: Augmenting Human Intellect*, pages 138–152. IOS Press, 2023.

[93] Y. Kubo, S. Karita, and M. Bacchiani. Knowledge transfer from large-scale pretrained language models to end-to-end speech recognizers. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8512–8516. IEEE, 2022.

[94] P. Ladefoged. The revised international phonetic alphabet. *Language*, 66(3):550–552, 1990.

[95] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1096, 2019.

[96] N. Le Roux and Y. Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural computation*, 20 (6):1631–1649, 2008.

[97] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[98] V. I. Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[99] C. Lewis-Beck and M. Lewis-Beck. *Applied regression: An introduction*, volume 22. Sage publications, 2015.

[100] B. Li, H. Peng, R. Sainju, J. Yang, L. Yang, Y. Liang, W. Jiang, B. Wang, H. Liu, and C. Ding. Detecting gender bias in transformer-based models: A case study on bert. *arXiv preprint arXiv:2110.15733*, 2021.

[101] M. Li, Z. Zhao, and C. Scheidegger. Visualizing neural networks with the grand tour. *Distill*, 5(3):e25, 2020.

[102] N. Liu, J. Wang, and Y. Gong. Deep self-organizing map for visual classification. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2015.

[103] S. M. Liu and J.-H. Chen. A multi-label classification based approach for sentiment classification. *Expert Systems with Applications*, 42(3): 1083–1093, 2015.

[104] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

[105] D. Lu and Q. Weng. A survey of image classification methods and techniques for improving classification performance. *International journal of Remote sensing*, 28(5):823–870, 2007.

[106] S. J. Luck. *An Introduction to the Event-Related Potential Technique*, volume 78, 3. MIT press, 2005. ISBN 0262122774.

[107] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[108] A. Maas, Q. V. Le, T. M. O'neil, O. Vinyals, P. Nguyen, and A. Y. Ng. Recurrent neural networks for noise reduction in robust asr. In *Conference of the International Speech Communication Association (Interspeech)*, 2012.

[109] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 30, page 3. Citeseer, 2013.

[110] C. R. Madhuri, G. Anuradha, and M. V. Pujitha. House price prediction using regression techniques: A comparative study. In *International Conference on Smart Structures and Systems (ICSSS)*, pages 1–5. IEEE, 2019.

[111] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, 2015.

[112] S. Makeig, J. Onton, et al. ERP features and EEG dynamics: an ICA perspective. In *Oxford Handbook of Event-Related Potential Components*, pages 51–87. Oxford, 2011.

[113] K. Maurer and T. Dierks. *Atlas of Brain Mapping: Topographic Mapping of EEG and Evoked Potentials.* Springer Science & Business Media, 2012.

[114] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th Python in Science Conference (SciPy)*, volume 8, pages 18–25. Citeseer, 2015.

[115] L. McInnes, J. Healy, N. Saul, and L. Grossberger. UMAP: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018. URL `https://github.com/lmcinnes/umap`.

[116] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2020.

[117] R. Meudec. tf-explain, 2021. URL `https://github.com/sicara/tf-explain`.

[118] A. Meyer, S. Ghosh, D. Schindele, M. Schostak, S. Stober, C. Hansen, and M. Rak. Uncertainty-aware temporal self-learning (UATS): Semi-supervised learning for segmentation of prostate zones and beyond. *Artificial Intelligence in Medicine*, 116:102073, 2021.

[119] S. A. Mohammed, S. Darrab, S. A. Noaman, and G. Saake. Analysis of breast cancer detection using different machine learning techniques. In *5th International Conference on Data Mining and Big Data (DMBD), Belgrade, Serbia, July 14–20*, pages 108–117. Springer, 2020.

[120] C. Molnar. *Interpretable Machine Learning*. Independently published, 2nd edition, 2022. URL `https://christophm.github.io/interpretable-ml-book`.

[121] G. Montavon. LRP tutorial, 2021. URL `https://git.tu-berlin.de/gmontavon/lrp-tutorial`.

[122] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[123] A. S. Morcos, M. Raghu, and S. Bengio. Insights on representational similarity in neural networks with canonical correlation. *arXiv preprint arXiv:1806.05759*, 2018.

[124] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, 20(14):5, 2015.

[125] M. Müller. *Fundamentals of music processing: Audio, analysis, algorithms, applications*, volume 5. Springer, 2015.

[126] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[127] T. Nagamine and N. Mesgarani. Understanding the representation and computation of multilayer perceptrons: A case study in speech recognition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2564–2573, 2017.

[128] T. Nagamine, M. L. Seltzer, and N. Mesgarani. Exploring how deep neural networks form phonemic categories. In *Conference of the International Speech Communication Association (Interspeech)*, 2015.

[129] T. Nagamine, M. L. Seltzer, and N. Mesgarani. On the role of nonlinear transformations in deep neural network acoustic models. In *Conference of the International Speech Communication Association (Interspeech)*, pages 803–807, 2016.

[130] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[131] A. B. Nassif, M. A. Talib, Q. Nasir, and F. M. Dakalbab. Machine learning for anomaly detection: A systematic review. *IEEE Access*, 9: 78658–78700, 2021.

[132] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.

[133] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems*, pages 3387–3395, 2016.

[134] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4467–4477, 2017.

[135] W. Nie, Y. Zhang, and A. Patel. A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3809–3818, 2018.

[136] H. J. Nussbaumer and H. J. Nussbaumer. *The fast Fourier transform.* Springer, 1981.

[137] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017.

[138] OpenCV. Open Source Computer Vision Library, 2015. URL `https://github.com/opencv/opencv-python`.

[139] J. Oruh, S. Viriri, and A. Adegun. Long short-term memory recurrent neural network for automatic speech recognition. *IEEE Access*, 10: 30069–30079, 2022.

[140] S. Osindero and G. E. Hinton. Modeling image patches with a directed hierarchy of markov random fields. In *Advances in Neural Information Processing Systems*, pages 1121–1128, 2008.

[141] PAIR-code. What-If tool, 2019. URL `https://pair-code.github.io/what-if-tool/`.

[142] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: an ASR corpus based on public domain audio books. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[143] H. Park, N. Das, R. Duggal, A. P. Wright, O. Shaikh, F. Hohman, and D. H. P. Chau. Neurocartography tool, 2021. URL `https://poloclub.github.io/neuro-cartography/`.

[144] H. Park, N. Das, R. Duggal, A. P. Wright, O. Shaikh, F. Hohman, and D. H. P. Chau. Neurocartography: Scalable automatic visual summarization of concepts in deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):813–823, 2021.

[145] T. Pearce, A. Brintrup, and J. Zhu. Understanding softmax confidence and uncertainty. *arXiv preprint arXiv:2106.04972*, 2021.

[146] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[147] P. Pedersen. The mel scale. *Journal of Music Theory*, 9(2):295–308, 1965.

[148] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[149] L. Perotin, R. Serizel, E. Vincent, and A. Guérin. CRNN-based multiple DoA estimation using acoustic intensity features for Ambisonics recordings. *IEEE Journal of Selected Topics in Signal Processing*, 13(1): 22–33, 2019.

[150] R. Rahaman et al. Uncertainty quantification and deep ensembles. In *Advances in Neural Information Processing Systems*, volume 34, pages 20063–20075, 2021.

[151] K. Rao, F. Peng, H. Sak, and F. Beaufays. Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4225–4229, 2015.

[152] A. D. Rasamoelina, F. Adjailia, and P. Sinčák. A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286. IEEE, 2020.

[153] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

[154] J. Riget and J. S. Vesterstrøm. A diversity-guided particle swarm optimizer - the ARPSO. *Dept. Comput. Sci., Univ. of Aarhus, Aarhus, Denmark, Tech. Rep*, 2:2002, 2002.

[155] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[156] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[157] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115 (3):211–252, 2015.

[158] L. Schubert, C. Olah, A. Mordvintsev, I. Johnson, A. Satyanarayan, et al. Tensorflow lucid., 2019. URL `https://github.com/tensorflow/lucid`.

[159] D. A. Schult. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference (SciPy)*. Citeseer, 2008.

[160] K. Schulz, L. Sixt, F. Tombari, and T. Landgraf. Restricting the flow: Information bottlenecks for attribution. In *International Conference on Learning Representations (ICLR)*, 2019.

[161] E. Sejdić, I. Djurović, and J. Jiang. Time–frequency feature representation using energy concentration: An overview of recent advances. *Digital signal processing*, 19(1):153–183, 2009.

[162] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization. In *IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.

[163] J. L. Semmlow and B. Griffel. *Biosignal and medical image processing*. CRC press, 2021.

[164] A. Senior, G. Heigold, M. Ranzato, and K. Yang. An empirical study of learning rates in deep neural networks for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6724–6728, 2013.

[165] V. H. Shah. Machine learning techniques for stock prediction. *Foundations of Machine Learning*, 1(1):6–12, 2007.

[166] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *IEEE International Conference on Evolutionary Computation Proceedings*, pages 69–73, 1998.

[167] J. E. Shoup. Phonological aspects of speech recognition. *Trends in speech recognition*, pages 125–138, 1980.

[168] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[169] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3145–3153. PMLR, 2017.

[170] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[171] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[172] L. Sixt, M. Granz, and T. Landgraf. When Explanations Lie: Why Many Modified BP Attributions Fail. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 9046–9057, 2020.

[173] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[174] F. Stahlberg. Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 69:343–418, 2020.

[175] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3319–3328. PMLR, 2017.

[176] L. Sweeney. Discrimination in online ad delivery. *arXiv preprint arXiv:1301.6822*, 2013.

[177] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[178] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[179] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[180] H. Tanaka. editdistance, 2013. URL `https://github.com/roy-ht/editdistance`.

[181] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. YFCC100M: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[182] E. Thuillier, H. Gamper, and I. J. Tashev. Spatial audio feature discovery with convolutional neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6797–6801, 2018.

[183] C. Tomani, D. Cremers, and F. Buettner. Parameterized temperature scaling for boosting the expressive power in post-hoc uncertainty calibration. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 555–569. Springer, 2022.

[184] G. Trigeorgis, F. Ringeval, R. Brueckner, E. Marchi, M. A. Nicolaou, B. Schuller, and S. Zafeiriou. Adieu features? end-to-end speech emotion recognition using a deep convolutional recurrent network. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5200–5204, 2016.

[185] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 66:2579–2605, 2008.

[186] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[187] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1096–1103, 2008.

[188] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17:395–416, 2007.

[189] H. Wang, Z. Wang, M. Du, F. Yang, Z. Zhang, S. Ding, P. Mardziel, and X. Hu. Score-CAM: Score-weighted visual explanations for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) workshops*, pages 24–25, 2020.

[190] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viégas, and J. Wilson. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, 26(1):56–65, 2019.

[191] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[192] M.-S. Yang. A survey of fuzzy clustering. *Mathematical and Computer modelling*, 18(11):1–16, 1993.

[193] K. Yao and G. Zweig. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. In *Conference of the International Speech Communication Association (Interspeech)*. ISCA, 2015.

[194] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[195] D. Yu and L. Deng. *Automatic speech recognition*, volume 1. Springer, 2016.

[196] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 818–833. Springer, 2014.

[197] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016.

E h r e n e r k l ä r u n g

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 12.02.2024

Valerie Krug