Automatic Recognition of Non-Verbal Acoustic Communication Events With Neural Networks

Applied to Infant Vocalizations and Chimpanzee Calls

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

Dissertation

zur Erlangung des akademischen Grades

Doctor Rerum Naturalium
(Dr. rer. nat.)

im Fachgebiet

Informatik

Vorgelegt

von Master of Science Franz Anders

geboren am 01.09.1988 in Jena

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Gerik Scheuermann, Universität Leipzig &
   Prof. Dr. Mario Hlawitschka, HTWK Leipzig
2. Univ.-Prof. Dr. Franz Pernkopf, TU Graz

Die Verleihung des akademischen Grades erfolgt mit Bestehen
der Verteidigung am 06.05.2022 mit dem Gesamtprädikat magna cum laude.

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialen oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den 13.10.2021

Franz Anders

Professional and Scientific Career

Personal data

Name	Franz Anders
Birth dates	01.09.1988, Jena, Germany
E-mail	franz.anders.fa@gmail.com

Professional experience

2017	Apinauten GmbH, Leipzig: Student assistant in software development
2014 – 2015	Regiocast Leipzig: Student assistant in audio production
2014	Rohde & Schwarz Headend Systems Berlin: Internship in software development
2011 – 2013	P3 Systems Stuttgart: Student assistant
2007 – 2010	Antenne Thüringen and Radio Top 40 Weimar: Traineeship in audio production and music editor.

Education

2017 – 2021	PhD student: Leipzig University of Applied Sciences, research team Laboratory for Biosignal Processing & Leipzig University.
2014 – 2017	Master of Science: Media Informatics, Leipzig University of Applied Sciences
2010 – 2013	Bachelor of Engineering: Audiovisual Media, Stuttgart Media University
2000 – 2007	Abitur: Christliches Gymnasium Jena

List of Publications

Peer-reviewed articles as first author

- F. Anders, M. Hlawitschka, and M. Fuchs. Comparison of artificial neural network types for infant vocalization classification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:54–67, 2020. https://doi.org/10.1109/TASLP.2020.3037414
- F. Anders, M. Hlawitschka, and M. Fuchs. Automatic classification of infant vocalization sequences with convolutional neural networks. *Elsevier Speech Communication*, 119:36–45, 2020. https://doi.org/10.1016/j.specom.2020.03.003
- F. Anders, A. K. Kalan, H. S. Kuehl, and M. Fuchs. Compensating class imbalance for acoustic chimpanzee detection with convolutional recurrent neural networks. *Ecological Informatics*, 65, 2021. https://doi.org/10.1016/j.ecoinf.2021.101423

Preprints as first author

- F. Anders, M. Hlawitschka, and M. Fuchs. Investigation of the assessment of infant vocalizations by laypersons. *arXiv preprint*, 2021. https://arxiv.org/abs/2108.09205

Presentations

- F. Anders, M. Hlawitschka, and M. Fuchs. Automatisierte Erkennung und Klassifizierung von Säuglingslauten in Audiosignalen. In *20. Nachwuchswissenschaftlerkonferenz - Book of Abstracts*, pages 94–97. Hochschule Merseburg, 2020
- B. Reichard, F. Schrump, F. Anders, C. Mönch, K. Bode, and M. Fuchs. Utilizing automatically estimated facial descriptors for pain detection during surgical interventions. In *BMT Biomedical Technology Conference VDE*, 2020

Acknowledgments

I want to thank Mario Hlawitschka and Gerik Scheuermann, who supervised this work. You gave me the chance to do this PhD and always had trust in me and my research.

I want to thank Mirco Fuchs, my mentor at the Laboratory for Biosignal Processing. Although your expertise was of great help, your emotional support was even more important. Praise is rare in this business, but you always had praise left when I needed it.

I want to thank my brother, Philipp. There was no day where you had the slightest doubt that I could do this and it pulled me through. If you wouldn't have drawn the short straw, you'd be doing this PhD right now. And you'd be better at it. You are the most important person in my life.

I want to thank the University of Applied Sciences for funding my research. It was a luxury focusing on research exclusively for four years thanks to this grant and would not have been possible without this support.

I want to thank the staff at the Laboratory for Biosignal Processing for all the unconditional support. Particularly Patrick Frenzel, who ultimately enabled my experiments through his infrastructure and GPU management.

I want to thank Benni, Johannes, and Bianca, my colleagues for three years. Working with you by my side made the process much more enjoyable.

I want to thank Hartwig Junge, my coach in scientific writing. 99% of what I know about scientific writing I learned from you, and you alone. Your knowledge was invaluable for each and every paper I wrote.

I want to thank my parents, Christoph and Astrid. It's safe to say that starting a PhD at 29 years old is a privilege, which few people have. You made that privilege possible by supporting me far longer than you needed to, even when I decided to change careers late.

I want to thank everyone in Schwarzpulver. Thursdays were my weekly highlight for the past four years. I'm sure that someday we will refer to this period as the "golden times".

I want to thank Schlicht & Ergreifend. Particularly August: I know you hate to hear it, but you're one of my role models.

I want to thank the Reutlingen Crew. I always had great times with you all, particularly in Mathon. I hope we manage to stay in contact somehow. This also includes Micha and Marion.

In no particular order: Leo, Alex, Marie Sophie, Lina, Martin, Anne, Jan, Kai, Sabrina, Dennis, Dusan, Maria, Erwin, Lea, Javiera, Oli, Frans.

Finally, I want to thank Marlene. You gave me more than I could ever give back. I'm currently unable to do justice to the praise you deserve.

Abstract

Non-verbal acoustic communication is of high importance to humans and animals: Infants use the voice as a primary communication tool, signaling distress through crying, joy through laughing, and acquiring speech capacity through babbling. Animals of all kinds employ acoustic communication, such as chimpanzees, which use pant-hoot vocalizations or drumming for long-distance communication.

Many applications require the assessment of such communication for a variety of analysis goals. Computational systems can support these areas through automatization of the assessment process. This is of particular importance in monitoring scenarios over large spatial and time scales, which are infeasible to perform manually. Examples for this include wildlife monitoring of animals and long-term pain assessment in infants.

Algorithms for sound recognition have traditionally been based on conventional machine learning approaches. In recent years, so-called representation learning approaches have gained increasing popularity. This particularly includes *deep learning* approaches that feed raw data to deep neural networks. However, there remain open challenges in applying these approaches to automatic recognition of non-verbal acoustic communication events: Deep learning usually requires large data sets, whereas data sets in non-verbal acoustic communication are usually small. Therefore, deep learning based algorithms require thorough optimization to be applied optimally.

The leading question of this thesis is: *How can we apply deep learning more effectively to automatic recognition of non-verbal acoustic communication events?* The target communication types were specifically (1) infant vocalizations and (2) chimpanzee long-distance calls.

This thesis comprises four studies that investigated aspects of this question:

Study (A) investigated the assessment of infant vocalizations by laypersons. The central goal was to derive an infant vocalization classification scheme based on the laypersons' perception. The study method was based on the Nijmegen Protocol, where participants rated vocalization recordings through various items, such as affective ratings and class labels. Results showed a strong association between valence ratings and class labels, which was used to derive a classification scheme.

Study (B) was a comparative study on various neural network types for the automatic classification of infant vocalizations. The goal was to determine the best performing network type among the currently most prevailing ones, while considering the influence of their architectural configuration. Results showed that convolutional neural networks

outperformed recurrent neural networks and that the choice of the frequency and time aggregation layer inside the network is the most important architectural choice.

Study (C) was a detailed investigation on computer vision-like convolutional neural networks for infant vocalization classification. The goal was to determine the most important architectural properties for increasing classification performance. Results confirmed the importance of the aggregation layer and additionally identified the input size of the fully-connected layers and the accumulated receptive field to be of major importance.

Study (D) was an investigation on compensating class imbalance for chimpanzee call detection in naturalistic long-term recordings. The goal was to determine which compensation method among a selected group improved performance the most for a deep learning system. Results showed that spectrogram denoising was most effective, while methods for compensating relative imbalance either retained or decreased performance.

The implication of this thesis' results is that deep learning systems can match or even surpass the performance of conventional machine learning approaches for automatic recognition of non-verbal acoustic communication events. However, optimizing components such as the network architecture is crucial in absence of large data sets. The algorithms presented in this thesis are applicable for solving specific tasks on recognition of infant and chimpanzee acoustic communication events with high accuracy. The general insights gained through the studies may aid future research on which parts of their pipelines to focus first.

Mathematical Notation

$\mathbb{A}$	A set.
$\mathbb{N}$ / $\mathbb{R}$ / $\mathbb{C}$	The set of natural numbers, real numbers, and complex numbers respectively.
$\{0, 1\}$	A set, containing elements 0 and 1.
$\{0, 1, \dots, n\}$	A set, containing integers from 0 to n.
$[0, 1]$	The range of all real numbers between 0 and 1.
$\{x^{(1)}, \dots, x^{(n)}\}$	A set of elements $x^{(i)}$, where (i) is the element index.
$\{x^{(i)}\}_{i=\{1,\dots,n\}}$	Abbreviation of the expression above.
$(x^{(1)}, \dots, x^{(n)})$	A tuple of elements $x^{(i)}$, where (i) is the element index. Can be abbreviated likewise.
x	A scalar, e.g. $x \in \mathbb{R}$. Uppercase italics might also be scalars, e.g. X.
$\boldsymbol{x}$	A column vector, e.g. $\boldsymbol{x} \in \mathbb{R}^n$, with elements $[x_1, \dots, x_n]^\top$.
x_i	Element i of vector $\boldsymbol{x}$, e.g. $x_i \in \mathbb{R}$.
$\boldsymbol{x}_{i:j}$	Elements i to j of $\boldsymbol{x}$, i.e. $[x_i, \dots, x_j]^\top$
$\langle \boldsymbol{x}, \boldsymbol{y} \rangle$	The inner product between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, i.e. $\boldsymbol{x}^\top \cdot \boldsymbol{y}$.
$\boldsymbol{X}$	A matrix, e.g. $\boldsymbol{X} \in \mathbb{R}^{n \times m}$.
$X_{i,j}$	Element i, j of matrix $\boldsymbol{X}$, e.g. $X_{i,j} \in \mathbb{R}$.
$\boldsymbol{X}_{i,:}$	Row i of matrix $\boldsymbol{X}$, e.g. $\boldsymbol{X}_{i,:} \in \mathbb{R}^n$.
$\boldsymbol{X}_{:,j}$	Column j of matrix $\boldsymbol{X}$, e.g. $\boldsymbol{X}_{:,j} \in \mathbb{R}^m$.
$f : \mathbb{A} \mapsto \mathbb{B}$	A function signature, indicating that function f maps from domain $\mathbb{A}$ to codomain $\mathbb{B}$
$f(x; \theta)$	Function f is parametrized by θ. Mathematically, both x and θ are arguments to the function. Semantically, θ is considered fixed, while x is considered variable.
x	A scalar random variable.
$\mathbf{x}$	A vector random variable.
$p_{\mathrm{x}}(x)$	The probability distribution associated with x.

Contents

Professional and Scientific Career iii

List of Publications v

Acknowledgments vii

Abstract ix

Mathematical Notation xii

Contents xiii

1 Introduction 1
- 1.1 Thesis topic . . . 1
- 1.2 Global thesis goal . . . 4
- 1.3 Overview over studies (A) – (D) . . . 5
- 1.4 Thesis structure . . . 6
- 1.5 Summary of major findings . . . 7

2 Foundations in Automatic Recognition of Acoustic Communication Events 9
- 2.1 Acoustic analysis of non-verbal acoustic communication events . . . 9
- 2.2 Basics in conventional supervised machine learning . . . 18
- 2.3 Artificial neural networks as predictive models . . . 33
- 2.4 Automatic sound recognition through machine learning . . . 44

3 State of Research 53
- 3.1 Related scientific competitions and communities . . . 54
- 3.2 Deep learning in ASR and CP . . . 55
- 3.3 States of research specific to studies (A) – (D) . . . 59

4 Study (A): Investigation of the Assessment of Infant Vocalizations by Laypersons 65
- 4.1 Study goal . . . 65
- 4.2 Materials and methods . . . 66

4.3 Results . . . 73
4.4 Discussion . . . 82

5 Study (B): Comparison of Neural Network Types for Automatic Classification of Infant Vocalizations 85
5.1 Study goal . . . 85
5.2 Materials and methods . . . 86
5.3 Results . . . 98
5.4 Discussion . . . 105

6 Study (C): Detailed Investigation of CNNs for Automatic Classification of Infant Vocalizations 107
6.1 Study goal . . . 107
6.2 Materials and methods . . . 108
6.3 Results . . . 114
6.4 Discussion . . . 120

7 Study (D): Compensating Class Imbalance for Acoustic Chimpanzee Detection With Convolutional Recurrent Neural Networks 123
7.1 Study goal . . . 123
7.2 Materials and methods . . . 124
7.3 Results . . . 138
7.4 Discussion . . . 146

8 Conclusion and Collected Discussion 149
8.1 Conclusion . . . 149
8.2 Collected discussion . . . 150
8.3 Outlook . . . 151

Bibliography 153

9 Appendix 171
9.1 Appendix for study (A) . . . 171
9.2 Appendix for study (B) . . . 172

List of Figures 177

List of Tables 178

Acronyms 179

CHAPTER 1

Introduction

1.1 Thesis topic

1.1.1 Non-verbal acoustic communication and paralinguistics

Acoustic communication is the communication of humans or animals through sound waves. It is among the most important communication channels for exchange of information. **Vocal communication** is acoustic communication where sound is produced through mechanisms similar to human vocal cords. When communicative sounds are produced through other mechanisms, this is referred to as **non-vocal acoustic communication**.

The importance of acoustic communication is most obvious in humans. They developed the most sophisticated type of vocal communication among all animals: **speech** (also known as **verbal communication**), which is *spoken* language, such as spoken German. Sign of the importance of speech for humans is the development of the pharynx in infants: The infant's vocal tract is build to allow breathing and swallowing simultaneously, at the price of only enabling rudimentary vocal articulation. As an infant ages, the pharynx changes to enable more complex articulation, however at the cost of becoming susceptible to choking on food. Consequently, evolution must have valued speech highly. [148, chapter 1][129, 133]

Non-verbal acoustic communication is any acoustic communication apart from spoken, human language. Many animal species employ this communication form with varying degrees of complexity, intends and mechanisms of sound production. Vocal communication is common: Wolves howl to identify themselves within their group [138], meerkats emit alarm calls to indicate the predator type [100], and Bird songs are particularly complex, coding a variety of information, such as species identification, sex, or mating behavior [33, 102]. An example for **non-verbal non-vocal acoustic communication** are cricket chirps, which are produced through file and scraper structures on the wings [45, 76].

Humans use **non-verbal vocal communication** as well. One example of this is modulation of verbal speech to express emotions, either consciously or unconsciously. Another example are sounds that are not specifically words, but carry communicative value nonetheless, such as in *laughing* or *crying*. Some authors refer to such sounds as *calls* in animals and humans [12, 76]. **Paralinguistics** is the research discipline on human non-verbal, vocal communication (and, depending on the definition, on written communication as well). [148, p. 9]

1.1.2 Acoustic communication in infants and chimpanzees

This thesis focused specifically on the non-verbal acoustic communication of two species: *human infant vocalizations* and *chimpanzee long-distance calls*.

The voice is among the most important communication tool for infants. Adult humans assess infant vocalizations in a variety of contexts, and with varying analysis goals. Infants use negative affective vocalizations to signal distress, e.g. *crying* or *fussing*. Parents assess crying intuitively for ensuring the infant's well being [16, chapter 1,2]. In pediatric wards, nurses assess crying as a proxy for pain intensity to employ appropriate pain management measures [104, chapter 5]. Infants experiment with their voice and produce increasingly complex sounds as they develop in acquisition of the speech capacity. Professionals in assessment of vocal development monitor the emergence of certain types of such *protophones*, such as vowels, squeals, consonants, various types of babbling etc. to identify developmental delays [21, 115, 122, 163].

For chimpanzees, acoustic communication is of overall less importance, as they primarily use visual and tactile communication [19]. However, their long-distance communication is acoustic. The *pant-hoot* is perhaps the most stereotypical chimpanzee vocalization. The exact purpose of this call is still under active research, where signaling of spacing between groups is hypothesized to be a probable function. They also employ *drumming*, where they hit buttresses of trees with their hands and feet [13].

There is a body of behavioral research that focuses on chimpanzee and human infant vocalizations in combination. Firstly, because of the obvious evolutionary proximity of both species, and secondly because they share certain vocalization types. For example, both species elicit crying. It is acoustically similar for both and is hypothesized to signal distress, particularly separation from the mother. [16, chapter 10]

1.1.3 Objective assessment of non-verbal acoustic communication

Most areas that apply assessment of acoustic communication require *consistency* within and across observers, particularly in professional and scientific areas. Let us return to the example of estimating infant pain by assessing crying intensity. In this case, it is desirable that the same crying leads to the same pain intensity estimation regardless of who performs the assessment. However, purely intuitive and subjective assessment most likely leads to inconsistent evaluations. Consequently, we need to systematize the process of acoustic communication assessment.

For an objective assessment of acoustic communication we employ specialized **assessment tools**. These comprise various components: Firstly, they contain a **classification scheme**, which in this context is a catalogue-like listing of acoustic communication event classes. Secondly, there is an instruction that associates the occurrence of these classes with certain meanings. Most assessment tools define their own classification schemes. However, there are also catalogues that exclusively list communication event classes for their own sake. Assessment tools might then reference these catalogues to associate classes with meanings.

Infant pain scales are an example for such assessment tools. They are used in pediatrics to increase the consistency in pain assessment. They list vocalization classes (focused on distress vocalizations) and associate their occurrence with pain scores [104, chapter 5]. For example, the *The Neonatal Infant Pain Scale* (NIPS) defines: *no crying* = pain score 0; *whimper, mild moaning* = 1; *vigorous cry, loud scream* = 2 [88]. An example for a catalogue-style vocalization classification scheme is the one by Buder et al. [21], which is focuses on *protophone* (babble-like) vocalizations in infants.

1.1.4 Automatic recognition of non-verbal acoustic communication events

Many applications in assessment of non-verbal acoustic communication can benefit from the support of computational systems that automatize the assessment process. This applies particularly to monitoring applications that observe large temporal or spatial scales, which is infeasible to be realized manually.

For example, long-term pain assessment in infants is of high importance for chronic pain management, however strongly time-demanding on the staff [104, chapter 5][192]. Wildlife monitoring of chimpanzees in natural habitats depends heavily on acoustic monitoring, as the dense rain forest restricts visibility [74]. However, even when manual assessment by humans is feasible, automatic assessment can act as an "objective second opinion".

Such automatic systems ideally recognize a set of discrete classes of acoustic communication events against the background. This means they operate at the level of classification schemes, as opposed to at the level of one specific analysis goal. In this way, an automatic recognition system can be employed for various analysis goals through subsequent analysis of recognized event classes. For example, an automatic vocalization assessment system for infants would first recognize "objective" classes such as crying, laughing, breathing etc. A system aimed at pain assessment might subsequently count cry-occurrences to derive a pain score. A system aimed at analyzing vocal development might discard crying and count occurrences of protophones.

1.1.5 Deep learning for automatic sound recognition

An algorithm performing sound recognition of any kind is referred to as performing **automatic sound recognition** (**ASR**). Consequently, recognition of acoustic non-verbal communication events is a specific subtype of ASR. **Computational paralinguistics**

(**CP**) is also a subdiscipline of ASR focused on automatic recognition of paralinguistic phenomena.

Machine learning (**ML**) has been the dominant approach to designing automatic sound recognition systems for a long time. However, in recent years there has been a fundamental shift in the type of ML approach taken, from **conventional ML** approaches to **representation learning** approaches.

Conventional approaches emphasize representing audio signals through hand-crafted features that encode high-level and often application specific signal information. These are input to simple models, such as logistic regression. Opposed to this, representation learning emphasizes the use of complex models that operate on raw audio representations. **Deep learning** is a particular type of representation learning that uses deep neural networks as models. [46, chapter 1]

This shift was inspired by the "deep learning boom" in computer vision. Here, deep learning produced impressive performance gains over conventional ML approaches. This boom was initiated in 2012, when Krizhevsky et al. [85] managed to significantly outperform conventional ML approaches through deep learning in an international competition on image classification. Through this, Krizhevsky et al. [85] popularized **convolutional neural network**s (**CNN**s). Since then, many areas adapted similar deep learning approaches. One of those areas was ASR.

1.2 Global thesis goal

While in computer vision the transition from conventional approaches to deep learning is rather advanced and well investigated, applying deep learning to ASR is still under active research. Particularly, there is little research on how to transfer deep learning onto non-verbal acoustic communication events.

One of the main challenges is that deep learning usually requires large data sets compared to conventional approaches, as more parts of the recognition pipeline are dynamically adapted onto the data. While it is easy to gather large quantities of image data, gathering and labeling acoustic data is more time expensive, which causes these data sets to be smaller [176, chapter 6]. Data sets in non-verbal acoustic communication are particularly small, as providing reference annotations requires expert knowledge, as well as specialized recording procedures and subjects [150]. Consequently, components of the representation learning pipeline need to be highly optimized to use the available data effectively.

I conducted the research for this thesis in 2017 – 2020. During this time, the meta question underlying most ASR research was: *How can we apply deep learning more effectively to automatic sound recognition?* My research was heavily influenced by this. Consequently, the thesis goal was answering aspects of this question, specifically aimed at non-verbal acoustic communication events. More specifically, it was aimed at human infant vocalizations and chimpanzee acoustic communication.

Let me summarize the global goal of this thesis as follows:

1. The general goal is to advance our understanding of objective assessment of non-verbal acoustic communication events.
2. The more specific goal is to advance our understanding of using deep learning more effectively for automatic classification of non-verbal acoustic communication events with neural networks, applied to infant vocalizations and chimpanzee calls.

1.3 Overview over studies (A) – (D)

I performed my research through **four individual studies**, all within the aforementioned context of the global research goal. These studies carry the short-hand codes (A), (B), (C), and (D). Studies had mostly separate research questions and goals, and partly build on each other. Each study was associated with a scientific paper, which was either published in an international research journal or as a preprint. Consequently, large parts of this thesis consist of the content of those studies. However, their content was modified to accommodate the studies' recontextualization in this thesis.

The studies were as follows:

1.3.1 Study (A): Investigation of the Assessment of Infant Vocalizations by Laypersons

The goal of this study was to investigate the assessment capability of laypersons for infant vocalizations, to derive an infant vocalization classification scheme based on this assessment capability. This particular subject has not been investigated yet, as already existing classification schemes aim at trained professionals in vocalization assessment. The core knowledge gain was discovering the association between affective ratings and class labels for infant vocalizations, and the subsequent proposal of a classification scheme.

Fundamentally, this study investigated human perception and *not* automatic recognition algorithms. Primarily, this research helped advancing the general knowledge on objective assessment of infant vocalizations. However, it was also relevant for the development of automatic recognition algorithms: These require labeled acoustic data for training and validation of algorithms. Employing laypersons to label such data is the most time and cost efficient way to acquire large data sets. However, this requires knowledge on the laypersons assessment behavior, which was investigated in this study.

This study was published as a preprint, see [10].

1.3.2 Study (B): Comparison of Neural Network Types for Automatic Classification of Infant Vocalizations

The goal of this study was to determine the neural network type among the currently most prevalent ones with the best performance for infant vocalization classification, while also considering the influence of their architectural configuration. This is of importance, as the neural network architecture is the core module in a deep learning pipeline and thus of particular importance for performance optimization. Previous research worked

with fixed neural network architectures, leaving it unclear which aspects of the network architecture are of general importance for increasing performance. The core knowledge gain was the identification of those network stages and components with the highest relevance.

This study was published in the scientific journal *IEEE/ACM Transactions on Audio Speech and Language Processing*, see [9].

1.3.3 Study (C): Detailed Investigation of CNNs for Automatic Classification of Infant Vocalizations

The goal of this study was to identify the most important architectural properties of computer vision-like CNNs for automatic classification of infant vocalizations. This study was similar to study (B), however focused specifically on CNNs, instead of performing a more broad analysis on various network types. This is of importance, as computer-vision like CNNs are currently the most prevalent network type employed in ASR classification tasks. The core knowledge gain was the identification of the most relevant network properties.

This study partially builds on results of studies (A) and (B): The target classes resulted from the classification scheme developed in study (A). The investigated network type was among the most performant ones in study (B).

This study was published in the scientific journal *Elsevier Speech Communication*, see [9].

1.3.4 Study (D): Compensating Class Imbalance for Acoustic Chimpanzee Detection With Convolutional Recurrent Neural Networks

The goal of this study was to evaluate methods for compensating class imbalance for detection of chimpanzee calls in long-term recordings through deep learning. This was of importance, as chimpanzee calls are particularly rare in naturalistic monitoring scenarios. The resulting class imbalance is known to negatively affect predictive performance. Previous research circumvented this issue through artificial manipulation of the class imbalance in their data sets. The core knowledge gain was the identification of the most effective compensation methods for this application.

This study was published in the scientific journal *Elsevier Ecological Informatics*, see [11].

1.4 Thesis structure

This thesis is structured as follows:

Chapter 2 presents foundations and background knowledge for automatic classification of non-verbal acoustic communication events. It covers fundamental concepts that

are expected to be known to the reader to comprehend this thesis' studies. This comprises topics such as machine learning basics or neural network layer types. The chapter contains established knowledge in research on ASR and CP. Readers already familiar with those topics are free to skip this chapter.

Chapter 3 presents the state of research relevant to this thesis' studies and specifies the respective research gaps. It first summarizes relevant scientific competitions and their general developments. It then reports the specific state of research in (1) classification schemes for infant vocalizations, (2) automatic classification of infant vocalizations, and (3) automatic detection of chimpanzee calls, and highlights the research gaps in these areas.

Chapters 4, 5, 6, 7 present studies (A), (B), (C) and (D), respectively. The structure inside these chapters follows the common structure of scientific papers, i.e. statement of the study goal, materials and methods, results, and discussion.

Finally, chapter 8 presents a collected discussion of the findings and concludes the thesis.

1.5 Summary of major findings

For study (A), the major findings were as follows:

- Development of a classification scheme for infant vocalizations based on layperson perception. This scheme might be applied for large-scale labeling of acoustic data.
- The affective dimension *valence* has the overall highest association with acoustic class labels. This association was particularly strong for negative vocalizations, i.e. whining, crying and screaming, to the point of both being almost redundant. This implies that including valence ratings in labeling of acoustic data for infant vocalizations with laypersons is worthwhile. It also confirms the hypothesis of *crying as a graded signal* [16, chapter 2].
- Laypersons differentiated 9 salient acoustic classes. Consequently, they differentiate between fewer classes than commonly defined in classification schemes for infant vocal development, particularly regarding protophones.

For studies (A) – (D), the major findings were as follows:

- Deep learning systems managed to outperform conventional methods for infant vocalization recognition despite small data sets, if their architecture was optimized accordingly. This shows that optimizing the network architecture in deep learning systems is a worthwhile endeavor to significantly increase performance.
- The overall most important network architecture choice was the choice of the aggregation layer for reducing tensor dimensionality inside the network, e.g. choosing between a GAP layer or a flattening layer. Global pooling operations worked best on average. This implies that this architectural choice should be considered early in the network design process.

- Interaction effects played a significant role when analyzing associations between network properties and their performance. For example, the results of study (C) showed that the best values for the pooling size of CNNs interact with the choice of the aggregation layer. This implies that analysis methods should be chosen to include such interactions.
- Regarding the prior finding, regression trees proved to be an effective analysis tool to identify associations between network properties and performance.
- Convolutional neural networks outperformed pure recurrent neural networks for automatic classification with spectrogram inputs, regardless of whether they were combined with fully-connected or recurrent layers.
- The most important architectural properties in computer vision-like CNNs for infant vocalization classification, besides the aforementioned aggregation layer, were the cumulative receptive field size and the input size of the fully-connected layers.
- Study (D) showed that methods for compensating absolute class rarity were most effective for increasing performance to detect rare chimpanzee calls. Spectrogram denoising improved performance by several degrees. However, measures for compensating relative class imbalance retained or decreased performance, such as resampling of the data sets. This implies that teaching the neural network to decorrelate target class from background class characteristics is of primary importance to detect rare animal calls.

CHAPTER 2

Foundations in Automatic Recognition of Acoustic Communication Events

The goal of this chapter is to introduce basic concepts and terminology in automatic recognition of acoustic non-verbal communication events. This knowledge is a prerequisite for comprehending the relevant state of research and the thesis's studies.

There is no single textbook that comprises all of the required knowledge, as this thesis' research is interdisciplinary. Consequently, my contribution is the compilation of the knowledge from various sources into a concise introduction to the topic. The main sources used in this chapter are: [46, 69, 76, 148, 176]

The structure of this chapter is as follows. Section 2.1 introduces basic knowledge in acoustic analysis of non-verbal communication events. This is of importance to understand the nature of the studies' recognition tasks themselves, as well as the data sets. Section 2.2 introduces basic knowledge in machine learning. This is of importance, as ML is the basis for all current approaches to automatic sound recognition, including the approaches presented in the studies (B) – (D), as well as some of the analysis methods. Section 2.3 introduces basics on neural networks. This is of importance, as all of the studies' recognition algorithms involved neural network based approaches. Finally, section 2.4 introduces basics in automatic sound recognition.

2.1 Acoustic analysis of non-verbal acoustic communication events

Section 1.1.3 argued that any assessment of non-verbal acoustic communication behavior is based on a prior analysis off the occurrence of certain defined classes of communication events.

This is related to the distinction of the **form** and **function** of a concept, which is present in many research areas. Roughly speaking, form means "what does it look like" and function means "what is it used for" [148, chapter 1]. For example, imagine we aim to study infant crying, where the study goal is to analyze the meaning, i.e. the *function* of infant crying. To perform such an investigation using quantitative methods, we require a precise definition of how this vocalization type is defined acoustically, i.e. a definition on the *form* of crying. This ensures that the object of analysis and any subsequent analysis method is comprehensible and reproducible. Then, we might correlate the occurrence *cry* events and features with external factors to infer the meaning. Of course, the analysis goal might be anything apart from function-related analysis.

The goal of this chapter is to introduce basic methods and concepts on how to (1) define and discover classes of non-verbal acoustic communication events, and (2) how to identify such classes in recordings, once defined.

The structure of this section is as follows. Section 2.1.1 introduces spectrogram visualizations of audio signals. Spectrogram visualization are a basic tool for acoustic analysis. Additionally, they are used extensively in modern deep learning based ASR systems. Section 2.1.2 introduces an approach for discovering and extracting acoustic communication events from audio recordings. This is of importance for an understanding of the nature of the recognition tasks in all studies.

2.1.1 Spectrogram representations of audio signals

One of the most important tools for analysis of non-verbal acoustic communication events is audio signal visualization, e.g. to identify acoustic patterns of communication events. There exists a multitude of visualizations, all of which are based on certain signal transformations.

2.1.1.1 Waveforms

The continuous audio signal is a function $x_{\text{cont}}(t) = y$, where $t \in \mathbb{R}$ is a time point and $y \in \mathbb{R}$ is the associated amplitude. For computational processing, the continuous signal is digitized through sampling with a sampling frequency $F_s \in \mathbb{N}$. The resulting **digital signal** is a function $x(n) = y$, where $n \in \{0, \dots, L_{\text{sig}} - 1\} \subset \mathbb{N}$ is the sample index and $L_{\text{sig}} \in \mathbb{N}$ is the signal length. This signal is referred to as the **time domain signal**, as the input variable represents time. [176, chapter 4][63, chapter 5][44, chapter 2]

Visualization of an audio signal through amplitude over time is referred to as a **waveform**. This visualization indicates the time points of audio events of relative gains in signal amplitude, i.e. "loud" events. Waveforms indicate *when sounds are happening*, but are uninformative of sound characteristics. [176, chapter 4][63, chapter 5][44, chapter 2]

2.1.1.2 Frequency spectrum types

The human auditory system primarily judges the qualitative characteristics of sounds based on the **frequency spectrum**: The spectrum is the decomposition of a function into a set of elementary functions. These elementary functions are sinusoids of varying frequency, phase and amplitude. There are various transforms that define such as decomposition. The **discrete Fourier transform** (**DFT**) is one of the most fundamental ones and is pointwise defined as

$$\begin{aligned} DFT &: \mathbb{N} \mapsto \mathbb{C} \\ DFT(x)(f) = \tilde{x}(f) &= \sum_{k=0}^{N-1} x(k) e^{\frac{-i2\pi kf}{N}}, \end{aligned} \tag{2.1}$$

where $f \in \{0, \ldots, N/2\} \subset \mathbb{N}$ is the frequency index and $N \in \mathbb{N} \wedge N \geq L_{\text{sig}}$ is the STFT length. The result $\{\tilde{x}(f)\}_{f \in \{0,\ldots,N/2\}}$ is a frequency spectrum, where $\mathcal{R}(\tilde{x}(f))$ and $\mathcal{I}(\tilde{x}(f))$ indicate the amplitude of a cosine and sine at frequency f, respectively. It is also referred to as the **frequency domain signal** as the function input domain represents frequency.

The human auditory system evaluates the spectrum as the primary means to discriminate sounds. It is able to perceive frequencies in the range ≈ 0 – $20\,\text{kHz}$. [161, chapter 9]

However, direct visualization of the DFT is inconvenient due to the complex valued codomain. Therefore, we transform the spectrum into a purely real valued codomain. The most common variants are the **magnitude spectrum** $\tilde{x}_{\text{mag}}$ and **power spectrum** $\tilde{x}_{\text{power}}$, which are pointwise defined as

$$\begin{aligned} \tilde{x}_{\text{mag}} &: \mathbb{N} \mapsto \mathbb{R}, \quad \tilde{x}_{\text{mag}}(f) = |\tilde{x}(f)| \\ \tilde{x}_{\text{power}} &: \mathbb{N} \mapsto \mathbb{R}, \quad \tilde{x}_{\text{power}}(f) = \tilde{x}(f)^2. \end{aligned} \tag{2.2}$$

These transformations are also perceptually motivated. The human auditory system primarily evaluates the summed energy per frequency channel for judging sound characteristics, rather than evaluating the precise distribution of sines and cosines. [161, chapter 9][176, chapters 3]

For spectrum visualization, we plot magnitude or power against frequency. This visualization is more informative than the waveform regarding sound characteristics. It shows how the signal energy is distributed across frequency channels.

2.1.1.3 Spectrograms

The disadvantage of spectrum representations is that they dismiss temporal information. They represent the average distribution of frequencies across the analyzed time period. However, sound characteristics are not static, but change over time.

To combat this disadvantage, we segment the audio signal into **frames** of length $L_{\text{frame}} \in \mathbb{N} \wedge L_{\text{frame}} \leq L_{\text{sig}}$. This segmentation is characterized by the **frame length** and **hop size**, i.e. the duration between frames. Frames are so short that their signal

content is considered quasi stationary. Typical frame lengths are 20 – 40 ms. We calculate spectra for all frames and concatenate the results. This allows to analyze the temporal progression of sound characteristic. [148, chapter 7][176, chapters 4]

Transformations that incorporate time and frequency information are referred to as **time-frequency transformations**. Mathematically, they are defined pointwise through functions $S(t, f) = y$, where $t \in \{0, \dots, T-1\}$ is the time frame index (the start time of a frame), $f \in \{0, \dots, F-1\}$ is the frequency index, and y is the indexed value, such as the frequency magnitude or power.

Visualizations of time-frequency transformations are referred to as **spectrograms**. Spectrograms are 2D images, where the x-axis encodes time t, y-axis encodes frequency f, and pixel intensity or color encodes y. There are various spectrogram types, depending on the definition of the transformation. As an image is usually defined as a matrix, we collect all possible transformation values into a grid to construct such a matrix $\boldsymbol{S} \in \mathbb{R}^{T \times F}$ corresponding to the transformation as

$$\boldsymbol{S} = \begin{bmatrix} S(t=0, f=F) & \dots & S(t=T, t=F) \\ \dots & \dots & \dots \\ S(t=0, f=0) & \dots & S(t=T, f=0). \end{bmatrix} \tag{2.3}$$

For the remainder of this thesis, a spectrogram type's name is synonymous to its respective transformation, as the corresponding image matrix can always be derived according to equation 2.3. [176, chapter 4] [44, chapter 2]

Most spectrograms are based on the **short-time Fourier transform** (STFT), which implements the above described approach of performing DFTs on signal frames. It is defined pointwise as

$$\begin{gathered} S_{\text{comp}} : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{C} \\ STFT(x)(t, f) = S_{\text{comp}}(t, f) = \sum_{k=0}^{N-1} w(k) \cdot x(t \cdot N + H) \cdot e^{\frac{-i2\pi kf}{N}}, \end{gathered} \tag{2.4}$$

where $H \in \mathbb{N}$ is the hop size between frames, and N and f are defined according to equation 2.1.

The **window function** $w : \mathbb{R} \mapsto [0, 1]$ attenuates the distortion effects caused by the assumption of the DFT that the input frame is periodic. It also determines the frame length L_{frame} through its support. The **hanning window** is one of the most common window functions [130], which is defined as

$$w(n) = 0.5 \cdot (1 - \cos(\frac{2 \cdot \pi n}{L_{\text{frame}}})). \tag{2.5}$$

We transform the complex-valued STFT into variants with real-valued codomains just as the spectrum. The **magnitude spectrogram** and **power spectrogram** are defined, respectively, as

$$\begin{aligned} S_{\text{mag}} &: \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R}, \quad S_{\text{mag}}(t,f) = |S_{\text{comp}}(t,f)| \\ S_{\text{power}} &: \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R}, \quad S_{\text{power}}(t,f) = S_{\text{comp}}(t,f)^2. \end{aligned} \tag{2.6}$$

Either of these spectrogram variants might be used for visualization as a 2D image as described above.

However, the magnitude or power spectrogram does *not* translate directly to the human sound perception, although it is a close estimation. For example, humans do not perceive amplitudes linearly, i.e. a sound with twice as much energy is not necessarily perceived twice as loud. **Psycho acoustics** is the science on human sound perception, however the details of this are beyond the scope of this thesis. There are various techniques to process spectra to resemble the auditory perception more closely. Those with relevance to this thesis's studies are presented here.

The **log spectrogram** applies a point-wise logarithmic transformation to a magnitude or power spectrogram. This approximates the human perception of amplitudes more closely. It is defined as

$$\begin{aligned} S_{\log} &: \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R} \\ S_{\log}(t,f) &= \log(S(t,f) + \epsilon), \end{aligned} \tag{2.7}$$

where $\epsilon \approx 0$ is a small constant avoid calculating $log(0)$. The log spectrogram is among the most commonly used visualizations for research in acoustic communication in humans and animals, such as speech processing. [176, chapter3, 4][63, chapter 5][44, chapter 2]

The Mel-scaled spectrogram applies a non-linear transformation to the frequency axis to better approximate human perception, which is approximately logarithmic. It involves mapping of the linear frequencies to the **mel scale**. One of the most common definitions of the Mel scale is [176, chapter 4]

$$mel(f) = \frac{1000}{\log 2} \log(1 + \frac{f}{1000}). \tag{2.8}$$

The goal is to map linear frequencies of the spectrogram onto the Mel scale. For this, we construct a filter bank with M filters, where each filter $m(f)^{(i)} : \mathbb{R} \mapsto [0,1]$ has a triangular shape. We choose all filters to have the same width in the Mel scale and locate them to cover the entire available Mel scale. Figure 2.1 shows an exemplary Mel filter bank. We correlate the magnitude or power with this filter bank to receive the **Mel-scaled spectrogram** [176, chapter 4]

$$S_{\log \text{ Mel}}(f_{mel}) = \sum_{f=0}^{F-1} m^{(f_{mel})}(f) \cdot S(f), \tag{2.9}$$

where $f_{mel} \in \{1, \ldots, M\}$ is the filter index. Finally, we log-transform the Mel-scaled spectrogram as defined in Eq. 2.7 to receive the **log Mel-scaled** spectrogram.

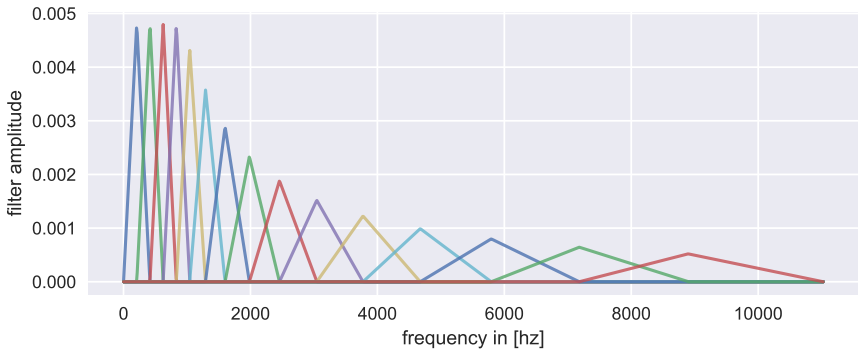

Figure 2.1: **Exemplary Mel filter bank.** In this example, there are $M = 15$ Mel filters. The filter bank calculation follows Slaney [158]. Here, filters are area-normalized.

There is not *the* Mel-spectrogram. There are various implementations for Mel scales and filterbanks that differ in details. Therefore, publications must state which one they used specifically. [176, chapter 4]

Figure 2.2 visualizes various spectrogram types presented in this section by the example on an input audio signal with infant crying. As shown, the audio wave form indicates time points of audio events, but is not indicative of the sound characteristics. The magnitude spectrum indicates the distribution of frequency components, but is not indicative of time points. The linear magnitude spectrogram appears to be empty, as the dynamic range is dominated by few values of high magnitude. The logarithmic spectrogram shows clearly visible patterns: Time points containing voice show wave patterns, while time points with pauses contain low intensities across the frequency range. Finally, the log Mel-scaled spectrogram changes the space between the wave patterns, increasing the resolution in low frequencies and decreasing it in high frequencies.

There are various, more sophisticated spectrogram variants that are not based on the STFT, such as wavelet transform [50]. However, discussion of such spectrograms is beyond the scope of this chapter.

2.1.2 Defining and extracting acoustic communication events

Two central issues in acoustic communication research are as follows. (1) **Discovery of classes**: Discovering abstract classes of acoustic communication events. (2) **Event extraction**: Given a set of such abstract class definitions, to detect event instances in an acoustic recording.

Kershenbaum et al. [76] presented a general framework for solving these tasks. Although the framework is specifically aimed at analysis of animal acoustic communication, I adapted the framework and its terminology for this thesis, as it is applicable to infant vocalizations as well. The remainder of this section is based on this paper.

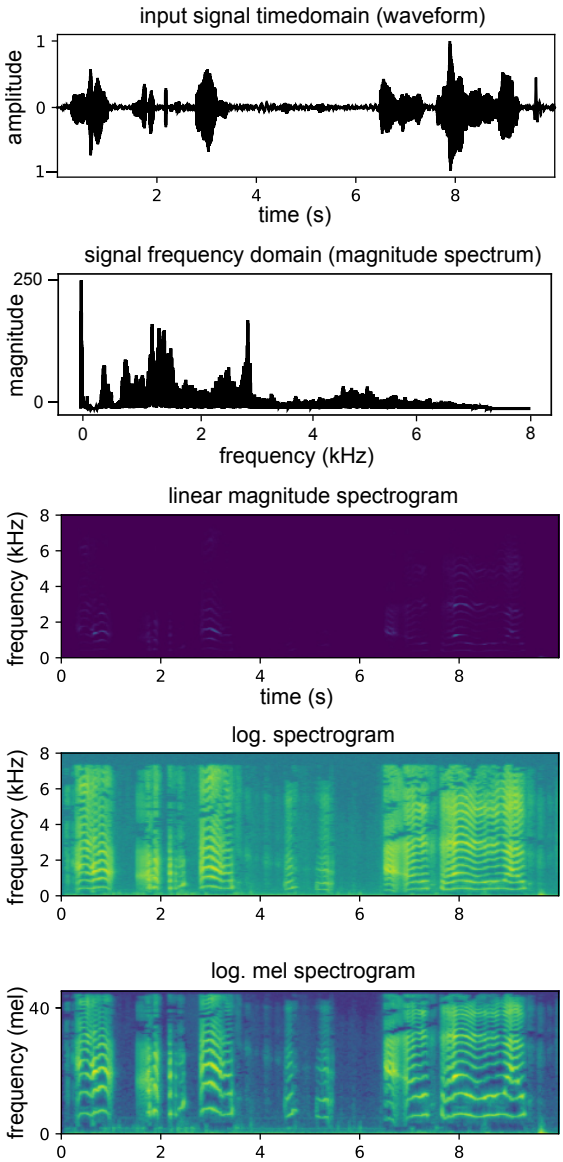

Figure 2.2: **Visualization of various audio representations.** The input audio signal at the top is an excerpt of an audio recording of an infant crying. Spectrograms visualize intensities through a color map, where dark colors indicate small intensities and light green colors indicate high intensities.

2.1.2.1 Discovery of classes

Given is a data set with recordings of the target species. The goal is to segment the recorded communication activity into smaller communication events. Kershenbaum et al. [76] refer to a communication event as an **unit**.

The segmentation of the communication activity into units is based on a **segmen-**

tation criterion. Application of these criteria involve inspection of the spectrogram. Figure 2.3 gives a schematic visualization of the most common criteria, which are defined as follows:

- **Criterion (A): Separation by silence.** Units are separated by pauses of a certain predetermined length.
- **Citerion (B): Change in acoustic properties (regardless of silence).** Units are separated by a significant change in acoustic property, regardless of silence. This segmentation criterion might be used to further subdivide units determined through criterion (A).
- **Criterion (C): Series of sounds.** Various sounds are grouped to a single unit if the pause between them is brief and they have similar spectrogram patterns.
- **Criterion (D): Higher levels or organizations.** Various sounds that might be considered different units according to criteria (A) – (C) might be grouped into a single unit, if they are discovered to always occur together.

The segmentation criterion must be determined by the researcher based on prior knowledge and hypothesis on the communication behavior of the species. A typical example of a unit is a *note* in bird songs.

A segmentation criterion that is common in research on infant vocal development specifically is the **breath group**. Nathani and Oller [116] defined it as "vocalizations or groups of vocalizations separated from all others by audible ingressive breaths or separable in accord with adult judges' intuitions that vocalizations are separated by pauses that potentially could include ingressive breaths". This segmentation criterion is most similar to segmentation criterion (A).

Discovering **unit classes** or **unit types** works as follows. First we group the extracted units into clusters of similar acoustic properties. Here, we have various degrees of freedom on (1) which properties we consider and (2) which grouping criterion we apply. We then derive general descriptions for the groups, which are then used as the abstract definitions of unit classes. From the point of view of machine learning, we perform clustering of units. In this context, a **classification scheme** is a list of such abstract class definitions. [76]

The discovery of unit classes implies the existence of an abstract parent type, of which classes are various subforms. I refer to this parent type as the **unit super type**. The central property of the unit super type is the segmentation criterion.

To summarize, there are three building blocks to units:

- The unit super type, which is the abstract definition of the building block type in acoustic communication that we analyze. Its central property is the expected length, defined through the segmentation criterion. An example for this is a *breath group*.
- Unit classes, which are different forms of the super type. Examples of this are *crying* and *laughing* as concepts.

- Units, which are actual instances of communication events located in a recording, delineated by the segmentation criterion. Examples of this are actual instances of *crying* and *laughing* in a recording. [148, chapter 3].[1]

Research disciplines that are specialized in certain species usually have a consensus on certain unit super types and classes. In many areas, the term **call** is synonymous to unit super types with segmentation criterion (A) – (C), for example in research on chimpanzees, killer whales or frogs [41, 76, 79, 118]. Some researchers adapted this term also for human non-verbal vocal communication for adults and infants [12, 145]. I highlight that the term **call type** in these papers corresponds to what I defined as unit classes.

A string of consecutive units might be grouped together to form a **sequence**. A typical example for a sequence is a *bird song*, consisting of various units of the super type *note*. We might also consider a *sentence* as a sequence of the unit super type *word* in linguistics. We can subject sequences to any further analysis. For example, we might cluster sequences to discover various sequence classes or correlate the occurrence of sequences with species behavior to discover their function. Essentially, sequences are higher levels of organization of units similar to the unit segmentation criterion (D).

2.1.2.2 Challenges in extraction of events

After the unit super type and classes have been formally defined, we can extract unit examples from recordings. The approach is as follows: (1) First we identify all units in the recording through application of the segmentation criterion. Each unit is defined through a start and end time point. (2) We label each unit according to the classification scheme, i.e. we label them according to the unit classes.

The main challenges in this are as follows.

(1) Ideally, the segmentation criterion is agnostic to the unit classes, so that we can first segment a recording and then label the units. However, in practice segmentation criteria and the unit classes can depend on each other. This is obvious from the segmentation criteria shown in Fig. 2.3. For example, criterion (D) requires identifying unit classes to know whether units are separated by silence or not. Another example of this is the unit type *word* in human speech, where identification of words in a sentence usually requires knowledge of the language, i.e. the set of word classes.

(2) Classification schemes can be limited to certain subtypes of acoustic communication of the target species, according to the communication type it was designed for. For example, Xie et al. [187] proposed an elaborate classification scheme where the unit super type is a *cry mode*, i.e. phoneme-like units inside cry vocalizations (similar to segmentation criterion (B)). Application of this scheme requires that the communication type itself is limited to cry vocalizations. However, determining whether this is the case is often not part of the classification scheme. [76][176, chapter 6][148, chapter 3]

[1] This differentiation expands on the original framework by Kershenbaum et al. [76], which only defined the terms *unit* and *label*. However, I expanded these definitions to differentiate these concepts further.

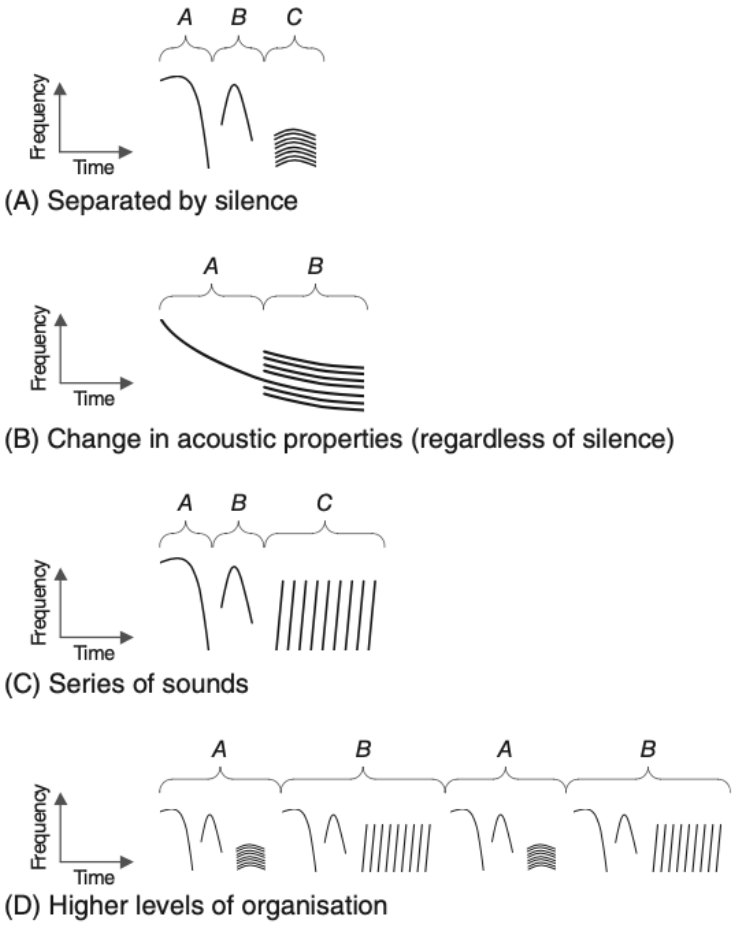

Figure 2.3: **Overview over the various segmentation criteria for units.** Image source: Kershenbaum et al. [76, p. 20]

2.2 Basics in conventional supervised machine learning

The goal of a recognition task is to predict information of interest about input data. For example, the input data might be an audio signal and the predicted information is the acoustic communication event class. The algorithm performing the mapping from input to output is referred to as the **model**. There are two approaches to designing such a model: The **knowledge based approach** and **machine learning** (**ML**). [46, chapter 1]

The knowledge based approach involves designing the model purely based on our assumptions. The model comprises of a set of *fixed* rules. *Fixed* in this context means that we, as the designers of the algorithm, determine the model rules based on our hypothesis on the relationship between input and prediction, and possibly empirical observations on the relationship. [46, chapter 1].

An example of a knowledge-based algorithm for infant vocalization recognition is the model of Mima and Arakawa [112]. They presented a model for automatic recognition of the crying cause (sleepiness, hunger, discomfort) based on the cry signal. The algorithm is a decision tree, which analyzes the spectrogram patterns (see section 2.1.1 on spectrograms). For example, according to the tree, vocalizations with high energies in the range of 0 – 10 kHz and a sharp drop off in energy above this range indicates hunger. The decision tree rules were based on observations and assumptions of the authors. [46, chapter 1]

ML is an alternative approach for designing such algorithms. Here, the general shape of the model is still fixed, i.e. the function family. However, the actual content of the model is determined through automatic optimization on the data. If Mima and Arakawa [112] had determined the rules of their decision tree through some algorithmic optimization process, instead of determining them themselves, they would have performed machine learning. [46, chapter 1]

Currently, virtually any modern algorithm for recognition tasks is based on ML, e.g. computer vision tasks, ASR tasks, or even predicting customer behavior on websites. [46, chapter 1]

The above described scenario is a particular kind of ML, referred to as **supervised machine learning** (**SML**). Supervised machine learning requires that input data is associated with output information of interest. There is also **unsupervised machine learning** (**USML**), where the task is to discover general patterns in data, without aiming at a particular output information.

The goal of this section is the introduction to the basics in SML. All of the studies' recognition algorithms as well as some of the analysis tools were based on SML. While study (A) employed some USML as well, this area was of overall lesser importance to this thesis. Consequently, any methods associated with USML are introduced in study (A)'s method section.

The structure of this section is as follows. Section 2.2.1 explains the basic framework and terminology of SML. Section 2.2.2 elaborates on core ideas of a model's ability to generalize. Section 2.2.3 introduces SML task types with relevance to this thesis's studies. Section 2.2.4 introduces methods for evaluation of model performance. Section 2.2.5 introduces a selected set of conventional machine learning models.

2.2.1 Core concepts of supervised machine learning

The common definition of machine learning from Mitchell et al. [114] is as follows:

"*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*"

The exact definitions of T, E and P depend on the application.

2.2.1.1 The task

SML assumes a scenario in which examples $\boldsymbol{x} \in \mathbb{X}$ are associated with targets $\boldsymbol{y} \in \mathbb{Y}$. This association is expressed through a function $f_{\text{true}}(\boldsymbol{x}) = \boldsymbol{y}$. I will leave domains abstract for the remainder of this introduction. We might imagine $\boldsymbol{x}$ as the recording of an infant and $\boldsymbol{y}$ as the predicted vocalization class. However, f_{true} is unknown. The goal is to to find a **model** $f : \mathbb{X} \mapsto \mathbb{Y}$ that performs a prediction $f(\boldsymbol{x}) = \hat{\boldsymbol{y}}$, so that $\hat{\boldsymbol{y}} \approx \boldsymbol{y}$. In the definition of Mitchell [114], the task T is the prediction $f(\boldsymbol{x}) = \hat{\boldsymbol{y}}$. [176, chapter 5][69, chapter 2][46, chapter 5]

2.2.1.2 Examples and targets

An **example** $\boldsymbol{x}$ can be of any domain $\mathbb{X}$, e.g. vectors, matrices etc. Typically, examples are vectors $\boldsymbol{x} = [x_1 \in \mathbb{F}^{(1)}, \ldots, x_N \in \mathbb{F}^{(N)}]^\top$, where each vector component *and* domain is called a **feature**, and $N \in \mathbb{N}$ is the amount of features[2]. $\boldsymbol{x}$ is then referred to as a **feature vector**. Features can have domains of varying levels of measurement, e.g. continuous or nominal. For the remainder of this introduction, we assume examples to contain exclusively continuous features, so that $\boldsymbol{x} \in \mathbb{R}^N$. [176, chapter 5][69, chapter 2][46, chapter 5]

The **target** $\boldsymbol{y}$ is also referred to as the **label** or **response**. The target domain $\mathbb{Y}$ depends on the specific task. The two most common tasks in SML are classification and regression. In **classification**, the target has a nominal level of measurement, i.e. it is an unordered set of classes $\mathbb{Y} = \{c^{(1)}, \ldots, c^{(K)}\}$. In **regression**, the target domain is continuous $\mathbb{Y} = \mathbb{R}$. For the remainder of this introduction we assume targets to be vectors $\boldsymbol{y} \in \mathbb{R}^K$. To better differentiate $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$, the true targets $\boldsymbol{y}$ are also referred to the **ground truth**, and $\hat{\boldsymbol{y}}$ is referred to as the **predicted** target. [176, chapter 5][69, chapter 2][46, chapter 5]

Supervised machine learning assumes that the examples and targets follow probability distributions $p_{\mathbf{x}} : \mathbb{X} \mapsto [0, 1]$ and $p_y : \mathbb{Y} \mapsto [0, 1]$, e.g. an example $\boldsymbol{x}$ has probability $p_{\mathbf{x}}(\boldsymbol{x})$. Both variables follow a joint probability distribution $p_{\mathbf{x},\mathbf{y}} : \mathbb{X} \times \mathbb{Y} \mapsto [0, 1]$. However, these probability distributions are unknown. [176, chapter 5]

2.2.1.3 The performance

The **performance** P measures the agreement between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ trough a function $L : \mathbb{Y} \times \mathbb{Y} \mapsto \mathbb{R}$. The function L is also referred to as a **cost**, **loss** or **error** function. The term *performance* usually implies that high values of L indicate high agreement, while cost / error / loss function implies that low values indicate high agreement. To determine the expected cost $J_{\text{real}} \in \mathbb{R}$ of f, ideally we would compute the expected loss over the entire data generating process as

$$J_{\text{real}} = \int p_{\mathbf{x},\mathbf{y}}(\boldsymbol{x}, \boldsymbol{y}) \cdot L(\boldsymbol{y}, f(\boldsymbol{x}))\ d\boldsymbol{x}\ d\boldsymbol{y}. \tag{2.10}$$

[2] As the term *feature* applies to both, the value and the domain, we have to infer from the context which of both is meant.

However, we do not have access to the data generating distribution. Instead, we have access to a finite **data set** $\mathbb{S}$, which is a collection of examples with associated true targets $\mathbb{S} = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \dots, (\boldsymbol{x}^{(M)}, \boldsymbol{y}^{(M)})\}$ sampled from the distribution $p_{\mathbf{x},\mathbf{y}}$, where $M \in \mathbb{N}$ is the amount of examples. This set is used to approximate the performance through the **empirical risk**, which is defined as [176, chapter 5][69, chapter 2][46, chapter 5]

$$J_{\text{real}} \approx J(\mathbb{S}) = \frac{1}{|\mathbb{S}|} \sum_{\boldsymbol{x},\boldsymbol{y} \in \mathbb{S}} L(\boldsymbol{y}, f(\boldsymbol{x})). \tag{2.11}$$

2.2.1.4 Hypothesis space and model weights

The goal is to choose f to minimize equation 2.11. Therefore, we provide f with variability through a set of **parameters** or **weights** $\boldsymbol{\theta} \in \Theta$, i.e. $f(\boldsymbol{x}; \boldsymbol{\theta}) = \hat{\boldsymbol{y}}$. We modulate f by modulating $\boldsymbol{\theta}$. The manner of how these parameters are applied to an input example is subject to the explicit function body of f. The space of functions that are possible solutions to f via choosing θ is referred to as the model's **hypothesis space**. For example, the hypothesis space of a linear model are all linear functions. [46, chapter 5]

2.2.1.5 Learning

Minimization of equation 2.11 is referred to as **empirical risk minimization**, which is defined as

$$\boldsymbol{\theta}^*(\mathbb{S}) = \text{argmin}_{\boldsymbol{\theta} \in \Theta} \frac{1}{|\mathbb{S}|} \sum_{\boldsymbol{x},\boldsymbol{y} \in \mathbb{S}} L(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta})), \tag{2.12}$$

where $\boldsymbol{\theta}^*$ is the optimal set of weights. The process of finding $\boldsymbol{\theta}^*$ is referred to as **fitting a model** or **learning**. An exhaustive evaluation of the entire space Θ is usually infeasible. Therefore, $\boldsymbol{\theta}^*$ is determined through an **optimization algorithm** or **learning algorithm**, which performs some form of more sophisticated evaluation of Θ. [176, chapter 5][46, chapter 8]

If the learning algorithm is based on calculating derivatives (which it is for neural networks), L needs to be differentiable. Particularly for classification loss functions, this might not be the case. In such a case we define a **surrogate loss function** $l : \mathbb{Y} \times \mathbb{Y} \mapsto \mathbb{R}$, so that $l \approx L$. We then replace l with L in any equation. [176, chapter 5][46, chapter 8]

2.2.1.6 Training, testing and generalization performance

Our goal is to apply f to new, unseen data outside of $\mathbb{S}$. The performance on such data is referred to as the **generalization performance**. However, optimization of equation 2.12 does not ensure good performance on unseen data: The model could simply memorize $\mathbb{S}$, yielding a perfect score for equation 2.12, without having any applicability for examples outside of $\mathbb{S}$. [69, chapter 2][46, chapter 5]

The general rule in ML is that a data set used for optimization of any component in the ML system is prone for overestimating the generalization performance, as it becomes prone to the memorization effect. [69, chapter 2][46, chapter 5]

To combat this issue, we split $\mathbb{S}$ into disjoint subsets, the development set $\mathbb{S}_{\text{dev}}$ and the test set $\mathbb{S}_{\text{test}}$. Performances measured on these sets are referred to as the **development performance** and **test performance**, respectively. [176, chapter 6][69, chapter 2]

The **development set** is used for training, i.e. input to equation 2.12 with $\boldsymbol{\theta}^*(\mathbb{S} = \mathbb{S}_{\text{dev}})$. Regarding the definition of Mitchell [114], the development set is the *experience* E of the model. Optimization using the development set is referred to as **training** the model. We might measure the resulting performance on the development set, however this will overestimate the generalization performance, as argued above. [176, chapter 5][69, chapter 2][46, chapter 5]

The test set is used for estimation of the generalization performance through equation 2.11, i.e. $J(\mathbb{S} = \mathbb{S}_{\text{test}}; \boldsymbol{\theta}^*)$. [176, chapter 5][69, chapter 2][46, chapter 5]

2.2.1.7 Validation and hyperparameter optimization

The goal is to increase the performance on the test set, while only being able to perform optimization of model weights $\boldsymbol{\theta}$ directly on the development set. To accomplish this goal, we might modify system components apart from $\boldsymbol{\theta}$, such as the form of the model itself.

Any choice about the ML system that lies outside of $\boldsymbol{\theta}$ is referred to as a **hyperparameter**. An alternative definition is that a hyperparameter is any choice about the system determined *before* optimization of $\boldsymbol{\theta}$.

Optimizing hyperparameters is referred to as **hyperparameter optimization** or **hyperparameter tuning**. For example, let us choose the optimal model f from a set of possible models $\mathbb{M}$ through optimization defined as

$$f^*(\mathbb{S}; \boldsymbol{\theta}^*) = \text{argmin}_{f \in \mathbb{M}} J(\mathbb{S}; \boldsymbol{\theta}^*, f). \tag{2.13}$$

The algorithm used for hyperparameter optimization is referred to as a **hyperparameter optimization algorithm**. The algorithm might be an exhaustive evaluation of a predetermined hyperparameter space, evaluation of a randomly drawn subsample of that space, or a more sophisticated algorithm such as genetic optimization. [176, chapter 5][46, chapter 11]

However, we must not perform hyperparameter optimization on the test set, as then it would be directly involved in optimization again, which could cause the memorization effect.

Consequently, we split the development into two disjoint subsets, the training set $\mathbb{S}_{\text{train}}$ and validation set $\mathbb{S}_{\text{val}}$. We use the training set for direct optimization of model weights through equation 2.12, i.e. $\boldsymbol{\theta}^*(\mathbb{S} = \mathbb{S}_{\text{train}})$. We use the validation set as an intermediate estimate of the generalization performance. We also use it for hyperparameter optimization through equation 2.13, i.e. $f^*(\mathbb{S} = \mathbb{S}_{\text{val}}; \boldsymbol{\theta}^*)$. Finally, we measure the performance on the remaining test set as a final estimate of the generalization performance

through equation 2.11, i.e. $J(\mathbb{S} = \mathbb{S}_{\text{test}}; \boldsymbol{\theta}^*, f^*)$. [176, chapter 5][69, chapter 2][46, chapter 5]

Some sources refer to the development set as the training set as well, i.e. the term *training set* is used for two concepts: The data set not used for final testing and the subset of this data set used for optimization of model weights. Here, we have to infer the exact meaning of the term *training set* from the context. However, this naming convention is advantageous when we do *not* aim to choose hyperparameters and therefore do not need a validation set. Here, the development set equals the training set. In these settings, all hyperparameters are fixed based on prior assumptions. This is commonly the case when the goal is to analyze the associations between features and targets, rather than optimization prediction performance, such as when applying linear regression to measure linear dependencies.

2.2.2 Increasing generalization capability of models

Generalization capability is the capability of a model to display high performance on new, unseen data. We estimate this performance using the test set, while optimizing the model itself exclusively on the training and validation set. This chapter introduces concepts and methods for increasing the generalization capability.

2.2.2.1 Core concepts on generalization: capacity, overfitting, underfitting

The more a data set is involved in optimization of ML algorithm components, the more its performance will overestimate the generalization performance. Therefore, the training performance will generally be higher then the test performance.

The gap between the training and the test error is referred to as the **generalization gap**. A model with a large generalization gap is referred to as **overfitting** the training data. Overfitting means that the training set error is attributed to memorization of examples, rather than the model having learned rules with general applicability. A small generalization gap, but with low training and test performance is referred to as **underfitting**. [46, chapter 5][69, chapter 2]

The **irreducible error** is the lower bound to the achievable test loss. Seldom is it possible to achieve a perfect test performance. The main reason for the irreducible error is **noise**, i.e. influences on the true relationship between examples and targets that are *unlearnable* by any model. Examples of this include (1) actual stochastic influences on the example-target-relationship, (2) features with influence on the relationship between examples and targets that have not been provided, (3) measurement errors of ground truth labels. The *human performance*, i.e. the performance of humans on a task, is closely related to the irreducible error. [46, chapter 5][69, chapter 2]

The ability of a model to overfit or underfit the data is largely determined by the model's **capacity** (alias **flexibility**). A model with high capacity is able to fit a variety of different functions. Such a model makes little assumptions about the relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$, i.e. it can model relationships of arbitrary complexity. A model with low capacity makes strong assumptions about the relationship $\boldsymbol{x}$ and $\boldsymbol{y}$ and is

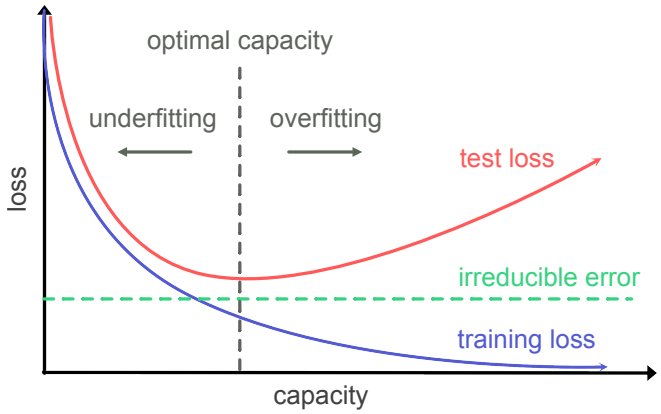

Figure 2.4: **Relationship between overfitting, underfitting and model capacity.**

unable to adapt when these assumptions are not met. In other words, models with high capacity have a large amount of functions in their hypothesis space, while models with low capacity have a small amount of specific functions in their hypothesis space. Usually (but not necessarily), models of low capacity assume simple relationships, such as linear relationships.

Figure 2.4 indicates the typical relationship between training loss and test loss as a function of the capacity. [46, chapter 5][69, chapter 2]

2.2.2.2 Aspects for reducing the generalization gap

The following list is a selection of areas that are commonly approached to reduce the generalization gap. All of these measures are classified as hyperparameter optimization. All of these were part of system design and research questions in studies (B) – (D):

(4) **Increasing the data amount**: The larger the data set, the more the data set is representative of the true underlying distribution. However, increasing the data amount usually is limited through practical feasibility. There are methods for artificially increasing the variability in the data set. One of those methods is **data augmentation**, which is the application of artificial computational perturbations to examples to inject variability that the model should be robust to. This simulates an increased data set size. [46, chapter 5,7][69, chapter 2]

(2) **Model selection**: Ideally, we choose a model with low capacity, but with just the right assumptions about f_{true}. If the model capacity is larger then necessary, it will display overfitting. If the model has low capacity and the wrong hypothesis space, it will display underfitting. [69, chapter 6]

(3) **Regularization**: Regularization is the modification of the learning algorithm to prefer certain solutions $\boldsymbol{\theta}^*$ for the same model f that increase the generalization performance. For example, we might reward weight vectors that focus on few features by modifying the performance metric during optimization. Regularization can reduce

overfitting of a models whose capacity is too large. However, excessive regularization can lead to underfitting. [46, chapter 4, 6]

(4) **Manipulation of example representation**: Examples are represented through features. If some features are unrelated to the response, the model will fit them regardless to increase training performance if it has sufficient capacity to do so. However, if features are absent from the example representation that are actually part of the true relationship f_{true}, this increases the irreducible error. [69, chapter 6]

2.2.2.3 Basic hyperparameter optimization algorithms

The simplest and most prevalent algorithms for hyperparameter optimization are **grid search** and **random search**.

In grid search, for each hyperparameter $\{h^{(1)}, h^{(2)}, \dots\}$ we determine ranges we aim to investigate $\{\mathbb{H}^{(1)}, \mathbb{H}^{(2)}, \dots\}$, based on prior assumptions. We build a grid of *all* possible combinations of hyperparameter values exhaustively, which is also referred to as the **search space**. We choose the setting that reached the highest validation performance. We might also include additional measures, such as preferring settings with greater computational efficiency. The advantage of this approach is its simplicity. The disadvantage is that the number of combinations grows exponentially with the amount of hyperparameters and that the search space needs to be finite. [46, chapter 11]

In random search, we only evaluate a randomly drawn subset of the grid. The advantage of this approach is that it is computationally less expensive: Usually, only a few of all investigated hyperparameters contribute significantly to the performance, which can be identified faster using random search. This method allows for infinite search spaces, as only a finite subset needs to be investigated. [46, chapter 11]

Both searches are often performed iteratively from broad to fine: We begin with an initial guess on the search space, analyze the results, and refine the search space to the most auspicious ranges based on this analysis. [46, chapter 11]

There are numerous more sophisticated hyperparameter optimization algorithms, such as bayesian optimizations. These algorithms usually contain series of *exploration* and *exploitation* steps. Exploration means collecting performance data of unknown regions of the search space and exploitation means focusing the search on the most promising regions. These algorithms however introduce their own set of secondary hyperparameters, which determine the behavior of the hyperparameter optimization algorithms itself. Therefore, some authors still recommend random search as the gold standard, as more sophisticated algorithms were found to offer only small performance gains over random search (and sometimes perform worse), while introducing extensive additional complexity to the search process. [3, 40, 46, 140]. Readers interested in this topic are recommended to read Feurer and Hutter [40].

2.2.3 Task types and target encoding

This section gives a more detailed introduction to task types relevant to this thesis, as well as encoding of target vectors.

2.2.3.1 Regression

For this thesis, only simple **regression** tasks were of interest. Here, the target domain comprises of continuous values, i.e. $\mathbb{Y} = \mathbb{R}$.

2.2.3.2 Classification

Classification tasks have nominal target domains $\mathbb{Y} = \{k^{(1)}, k^{(2)}, \ldots, k^{(K)}\}$, where K is the number of classes and upper script (i) the the class index.

There are two types of encoding targets: (1) **integer encoding**, in which the targets are encoded $y \in \mathbb{Y} = \{0, \ldots, K-1\}$, where y indicates activity of class $i+1$, and (2) **one hot encoding**, in which targets are encoded as vectors $y \in \mathbb{Y} = \{0, 1\}^K$, where $y_i = 1$ indicates presence and $y_i = 0$ absence of class i.

In **binary classification**, there are two classes $K = 2$, where only one can be chosen. This task usually employs target integer encoding. This task is most often associated with detection tasks, where there is one class of particular interest to be recognized as either present or absent. $y = 1$ indicates presence and $y = 0$ indicates absence of this class. Consequently, $y = 1$ is referred to as a *positive* and $y = 0$ as a *negative* or *background*.

In **multi-class classification**, there are at least three classes $K > 2$, where exactly one class must be chosen for any example. Targets might be encoded either through integer or one-hot encoding.

In **multi-label classification**, there are at least two classes $K \geq 2$, while any number of classes can be chosen simultaneously for any example. For this we employ one-hot encoded target vectors. This task is closely related to binary classification, so that we perform binary classification for presence vs. absence of each class separately. Accordingly, binary classification tasks can be viewed as multi-label classification with one class $K = 1$, introducing a second dummy class to represent absence. [69, chapter 4][176, chapter 5]

Some models do not output directly onto the target domain, but output class probabilities, i.e. for one-hot encoded target vectors they output $\hat{\boldsymbol{p}} \in \mathbb{P} = [0, 1]^K$. Here, $\hat{p}_i$ indicates the probability of class i being active. These models require post processing function to map these probabilities onto the output domain $\Gamma : \mathbb{P} \mapsto \mathbb{Y}$. This mapping is referred to as **binarization**.

Designing the binarization function is another hyperparameter. The most common settings are as follows.

In multi-class settings, we usually choose the most probable class as

$$\Gamma(\hat{\boldsymbol{p}}) = \text{argmax}_{i \in \{1,\ldots,K\}}(\hat{p}_i) = \hat{y}, \tag{2.14}$$

where y is the one-hot encoded target domain.

For multi-label and binary tasks, we require class-specific thresholds $\boldsymbol{\gamma} \in [0, 1]^K$, where γ_k is the threshold for class i. The thresholding is implemented as

$$\Gamma(p_i; \boldsymbol{\gamma}) = \begin{cases} 1 & \text{if } \hat{p}_i > \gamma_i \\ 0 & \text{else} \end{cases} = \hat{y}_i, \tag{2.15}$$

where y is the one-hot encoded target domain. The common, unbiased threshold choice is $\forall i : \gamma_i = 0.5$. [22]

2.2.4 Measurement of model performance

There are various aspects to measuring model performance: First, we require a metric for comparing ground truth to predicted targets, which is presented in section 2.2.4.1. Second, we require a evaluation setup, which is presented in section 2.2.4.6.

2.2.4.1 Selected evaluation metrics for classification

Performance metrics are implementations for evaluation of the empirical risk (see Eq. 2.11). However, most performance metrics are more sophisticated than just presenting implementations for the cost function L. While equation 2.11 weights all examples equally, some applications require over- and underweighting of certain examples. [176, chapter 6]

Consequently, here we define performance metrics as functions $P : \mathbb{Y}^M \times \mathbb{Y}^M \mapsto \mathbb{R}$, with $P(\mathbb{y}, \hat{\mathbb{y}})$, where $\mathbb{y} = \{y^{(1)}, \ldots, y^{(M)}\}$, $\hat{\mathbb{y}} = \{y^{(1)}, \ldots, \hat{y}^{(M)}\}$, and M is the amount of examples. [176, chapter 6]

This section is limited to metrics for classification tasks. We assume target vector integer encoding, i.e. metrics are defined either for binary or multi-class classification tasks. For multi-label tasks, we calculate metrics for each class individually, as though each class was its own binary classification task on class presence vs. absence. We then receive a performance score for each class, which we might average across classes to receive a single score, which is referred to as **macro averaging**.

2.2.4.2 Accuracy

This is the simplest performance metric for binary and multi-class classification tasks. It is defined as

$$P = \frac{1}{M} \sum_{i=1}^{M} I(y^{(i)}, \hat{y}^{(i)}), \tag{2.16}$$

where I is the **identity function** defined as

$$I(y, \hat{y}) = \begin{cases} 1 & \text{if } y = \hat{y} \\ 0 & \text{else.} \end{cases} \tag{2.17}$$

The codomain has the range $[0, 1]$, where 0 indicates no and 1 is perfect agreement between ground truth and predicted targets. [176, chapter 6][148, chapter 11]

2.2.4.3 Unweighted average recall (UAR), a.k.a. balanced accuracy

This is a performance metric for binary and multi-class classification tasks. Let $\mathbb{i}_k$ be the set of indices of examples belonging to class k defined as

$$\mathbb{i}_k := \{i \mid i \in 1, \ldots, M \wedge I(y, k) = 1\}. \tag{2.18}$$

Unweighted average recall is then defined as:

$$P = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|\mathbb{i}_k|} \sum_{i \in \mathbb{i}_k} I(y^{(i)}, \hat{y}^{(i)}) \tag{2.19}$$

Balanced accuracy is similar to accuracy. However, it accounts for class imbalance, i.e. settings where the relative amount of examples per class is uneven. Classes with few examples are overweighted and classes with many examples are underweighted. This metric is of importance when all classes are of equal importance to the application, regardless of their relative occurrence in the data set. It is the standard metric for classification tasks in computational paralinguistics. [148, chapter 11]

2.2.4.4 F1, precision and recall

These metrics are closely related evaluation metrics for binary classification settings. They rely on intermediate statistics. First, we define a specified identity function as

$$I_{a,b}(y, \hat{y}) = \begin{cases} 1 & \text{if } y = a \wedge \hat{y} = b \\ 0 & \text{else.} \end{cases} \tag{2.20}$$

$I_{1,1}(y, \hat{y}) = 1$ is referred to as a **true positive**, $I_{0,0}(y, \hat{y}) = 1$ is referred to as a **true negative**, $I_{0,1}(y, \hat{y}) = 1$ is referred to as a **false positive**, and $I_{1,0}(y, \hat{y}) = 1$ is referred to as a **false negative**.

Based on this identity function, we calculate the intermediate statistics **true positives** TP, **true negatives** TN, **false positives** FP, and **false negatives** FN as

$$\begin{aligned} TP &= \sum_{i=1}^{M} I_{1,1}(y^{(i)}, \hat{y}^{(i)}) \quad TN = \sum_{i=1}^{M} I_{0,0}(y^{(i)}, \hat{y}^{(i)}) \\ FP &= \sum_{i=1}^{M} I_{0,1}(y^{(i)}, \hat{y}^{(i)}) \quad FN = \sum_{i=1}^{M} I_{1,0}(y^{(i)}, \hat{y}^{(i)}). \end{aligned} \tag{2.21}$$

Based on these, we define the metrics **precision** PR and **recall** RC as

$$PR = \frac{TP}{TP + FP} \quad RC = \frac{TP}{TP + FN}. \tag{2.22}$$

The **F1** score is then defined as:

$$F1 = \frac{2 \cdot PR \cdot RC}{RP + RC}. \tag{2.23}$$

Precision, Recall and F1 originate from information retrieval tasks. We usually employ them in highly imbalanced binary classification settings, i.e. when there are much more negative than positive class examples $|\{I(y^{(i)}, 0)\}_{i=0,\dots,M}| \gg |\{I(y^{(i)}, 1)\}_{i=0,\dots,M}|$. These are "needle in a haystack" problems, where the positive class is the needle. Precision indicates the fraction of true positives among the ones predicted as positive. Recall indicates the fraction of true positives among the actually positive examples. F1 summarizes precision and recall through the harmonic mean. [176, chapter 6][148, chapter 11]

2.2.4.5 Average precision (AP)

Average precision is a metric for binary classification problems, where the model outputs class probabilities $\hat{p} \in [0, 1]$, instead of outputting directly onto the target domain $\hat{y} \in \{0, 1\}$. As mentioned in section 2.2.3, these models require output binarization. Average precision is a metric specialized in evaluating the total quality of predictions before binarization.

Let $TP_\gamma, FP_\gamma, \dots$ be the true positives, false positives etc. resulting from thresholding predicted probabilities at γ. We define AP as

$$AP = \frac{1}{TP + FN} \sum_{\gamma \in \mathbb{\Gamma}} PC_\gamma, \tag{2.24}$$

where $\mathbb{\Gamma}$ is the set of thresholds at which new positive predictions are accepted. Average precision can be interpreted as the average precision across all possible binarization thresholds. [176, chapter 6]

2.2.4.6 Evaluation procedures

To summarize the relevant aspects introduced in section 2.2.1, the development and evaluation of an ML system comprises of three stages, which are as follows:

1. **Training**, i.e. optimization of model parameters. Training performance is usually not measured as it is of no interest.
2. **Validation and hyperparameter optimization**. We measure validation performance as an intermediate estimate of the generalization performance. We also tune hyperparameters on the validation set.
3. **Testing**, i.e. we measure the test performance using the best found model parameters and hyperparameters. [176, chapter 6][69, chapter 5]

There arevarious approaches for implementing this three-stage process. Figure 2.5 visualizes the ones explained in this section.

The most straight-forward implementation is using a **singular train-val-test split**, i.e. splitting the data set into fixed subsets $\mathbb{S}_{\text{train}}$, $\mathbb{S}_{\text{val}}$ and $\mathbb{S}_{\text{test}}$ and using them for the above described approach. The advantage of this approach is its simplicity. The

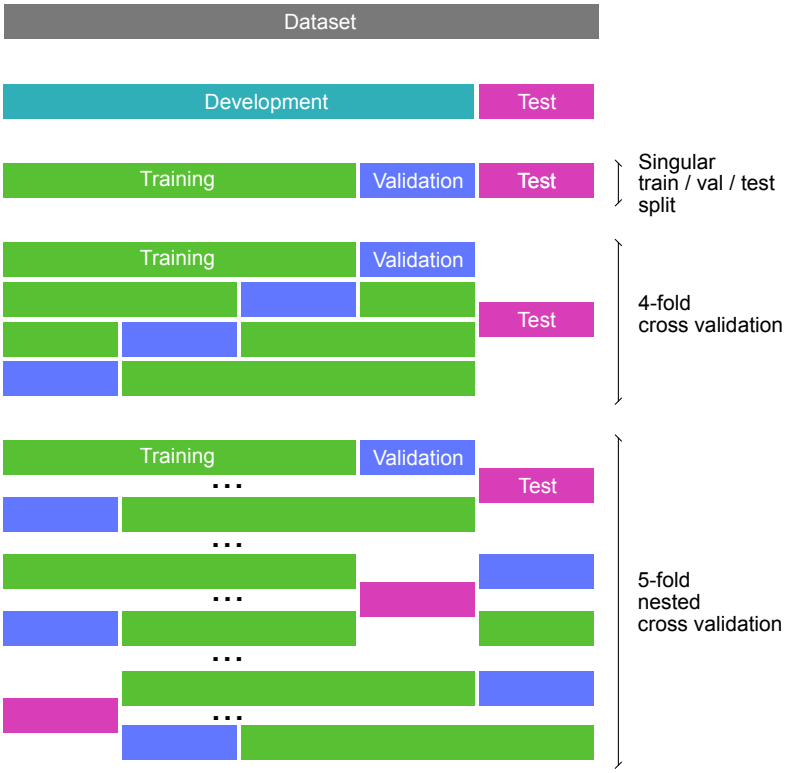

Figure 2.5: **Visualization of various evaluation procedures.**

disadvantage is that performance measures on the validation and test set are simple *point estimates*. Point estimates can be problematic for various reasons:

(1) **Random influences on the training algorithm**: Validation and test performance can vary between repeated experiments for the same hyperparameter setting if there are random influences on the training algorithm. An example for this is adding random noise to features for data augmentation (see section 2.2.2.2 on data augmentation).

(2) **Sampling bias**: Sampling bias means that a sample is not representative of the underlying distribution. Splitting the data can induce sampling bias when certain properties of examples become over- or underrepresented in subsets. For example, if by accident only the "difficult" examples were put into the test set, the performance will underestimate the generalization performance. The smaller the data set, the more prone it is to sampling bias. [176, chapter 6][69, chapter 5]

We combat issue (1) by repeating the training/validation/testing process multiple times on the *same* split. This way, we can derive the average performance of the hyperparameter setting based on the performance distribution. [176, chapter 6][69, chapter 5]

We combat issue (2) through **cross-validation**, i.e. repeating the training and validation process on *multiple* splits of the same data set. There are two variants of cross-

validation used in this thesis:

K-fold cross-validation subdivides the development set into $K \in \mathbb{N}$ mutually disjoint subsets $\mathbb{S}_{\text{fold}}^{(1)}, \ldots, \mathbb{S}_{\text{fold}}^{(K)}$, which are referred to as **folds**. Training and validation is repeated K times, where the n-th fold is used for validation and the remaining for training. This way, we receive K estimates of the validation performance to assess the mean and standard deviation. We then estimate the final generalization performance on the test set. K-fold cross-validation combats the sampling bias in the training and validation set, but not in the test set. [176, chapter 6][69, chapter 5]

Nested K-fold cross-validation is is an extension of this to combat sampling bias in the test set as well. Here, the entire data set is subdivided into K folds. Training, validation and testing is performed for each possible combination of folds, using one fold for validation, one for testing and the remaining for training. This way, we receive performance distributions for the validation and test performance and combat sampling bias in the test set as well. [179]

2.2.5 Selected machine learning models

This section presents selected machine learning algorithms with importance to this thesis' studies.

2.2.5.1 Multinominal logistic regression

Multinominal logistic regression is a model for multi-class classification tasks with K classes. It requires all features to be continuous. We assume one-hot encoded target vectors. It outputs probabilities, i.e. $f : \mathbb{R}^N \mapsto [0,1]^K$. The model is defined as

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) = g_{\text{softmax}}(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}), \tag{2.25}$$

where $\boldsymbol{x} \in \mathbb{R}^N$ is an input example, and $\boldsymbol{W} \in \mathbb{R}^K$ and $\boldsymbol{b} \in \mathbb{R}$ is the weight matrix and bias vector, respectively. Consequently, $\boldsymbol{W}_{i,:}$ and b_i are the linear weights and bias associated with class i, which perform a scalar product with the feature vector.

The function $g_{\text{softmax}} : \mathbb{R} \mapsto [0,1]$ is the **softmax function** defined as

$$g_{\text{softmax}}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}. \tag{2.26}$$

This function ensures that all output class probabilities sum to one. As this model outputs probabilities, it requires subsequent binarization to derive the final class (see section 2.2.3 on binarization).

2.2.5.2 Regression trees

Tree models build models that take on the form of binary trees. They aim at simple regression tasks (see section 2.2.3 on regression).

Each node in the tree model indicates *one* feature of the feature space $\mathbb{X}$. The respective feature is split into two regions, which are indicated by the branches leaving

the node. For continuous features, regions are separated by a threshold. For nominal features, regions are two mutual disjoint groups of values. Such splits are referred to as **axis-aligned cuts** or **axis-aligned splits**. Leafs indicate the predicted target value. To use a tree model for predicting new data, we follow the path from the tree root to the leaf by matching the input feature values to the tree splitting rules. [69, chapter 8][46, chapter 5]

At training time, the model is built through *greedy, recursive, binary splitting*. First, we select a loss function that we aim to minimize, such as the root mean square. To construct the first node, we test each possible axis-aligned cut of the feature space, i.e. we produce each possible split into two regions for each individual feature and calculate the respective loss improvement. We choose the one yielding the highest improvement on the training set. This approach is greedy, as we select the highest improvement one node at a time. It is recursive, as we repeat the process at each new node recursively. It is binary, as we only choose binary splits. [69, chapter 8][46, chapter 5]

This procedure leads to severe overfitting if we let the tree grow to arbitrary depth, as any cut would improve the target criterion on the training set. There are various methods to combat this. The most common one is to determine the optimal depth on unseen data on the validation set, possibly through cross-validation (see section 2.2.4.6 on cross-validation). Limiting the depth of a tree to avoid overfitting is referred to as **pruning**. [69, chapter 8][46, chapter 5]

Performance-wise, regression trees are usually outperformed by other, more sophisticated ML models. However, they have various advantages, which are as follows:

- **Mixed features**: Tree models are able to operate on mixed-type feature spaces, comprising nominal and continuous values without requiring dummy-encoding.
- **High interpretability**: Features and splitting rules are straight-forward to investigate for humans through visualization of the tree model.
- **Inherent feature selection and highlighting of feature importance**: Features with high overall importance occur near the tree root. Features with no importance are pruned off.
- **Consideration of feature interactions**: The recursive approach to finding splits can consider feature interactions. If a certain feature is globally unimportant, but of high importance inside a certain subgroup of the feature space, the tree can find it. Other models, such as straight linear models such as logistic regression, are unable to consider interactions. [69, chapter 8][46, chapter 5]

There are various implementations of regression trees, which differ in some aspects, such as the loss function or the pruning approach. Study (B) and (D) used regression trees for results analysis. The necessary details of their implementations are presented in the respective results sections, i.e. sections 5.3 and 7.3.

2.3 Artificial neural networks as predictive models

In the context of supervised machine learning as introduced in section 2.2, **artificial neural network** (**ANN**) are "simply" models for predicting $m : \mathbb{X} \mapsto \mathbb{Y}$. However, ANN have gained increasing popularity in recent years in the realm of recognition tasks for media data, namely image recognition tasks, sound recognition etc. At the time of writing this thesis, the majority of systems in these realms employs ANNs as the central model. For brevity, I refer to ANNs either as neural networks, or simply as networks in this thesis. [46]

The naming of ANN suggests a close relationship to biological neural networks. While biological neural networks might have been of higher importance in the early days of this technology, modern ANN departed from the biological model and are currently understood purely as mathematically defined models in the context of ML. Consequently, this section omits any discussion on biological parallels. [176, chapter 5]

Contrary to conventional ML models, neural networks offer immense degrees of freedom in their design. Their architecture is variable, where as conventional ML have fixed model designs. Even the most simplest class of neural networks allows for infinite architecture configurations. At its core, neural networks are more similar to a general framework on designing a certain class of ML models, than a specific model in and of itselfs. [46, chapters 5,6]

The goal of this chapter is to introduce the reader to the fundamentals in neural networks. This is of importance, as studies (B), (C) and (D) investigated neural networks applied to the automatic recognition of infant vocalizations and chimpanzee calls.

The structure of this section is as follows. Section 2.3.1 introduces the basic concepts of neural networks as layered models. Section 2.3.2 introduces fundamental layer types relevant to the neural networks applied in this thesis. Section 2.3.3 introduces basics on neural network training. Section 2.3.4 introduces architectures of particular neural networks relevant to this thesis.

2.3.1 Fundamental structure of an artificial neural network

A neural network is made up of $L \in \mathbb{N}$ layers, where each layer is some non-linear function. The functions are chained as a directed acyclic graph, so that each layer processes the result of its respective preceding layer. In a three layer network with $L = 3$, we construct the model as $f(\boldsymbol{x}) = l^{(3)}(l^{(2)}(l^{(1)}(\boldsymbol{x})))$. The upper script $(i) \in \{1, \ldots, L\}$ indexes the layer, e.g. $l^{(2)}$ is the second layer. We denote the output of the ith layer as $v^{(i)} \in \mathbb{V}^{(i)}$. Consequently, each layer defines a mapping $l^{(i)} : \mathbb{V}^{(i-1)} \mapsto \mathbb{V}^{(i)}$. Domains $\mathbb{V}^{(i)}$ are tensors of arbitrary dimensionality, i.e. $\mathbb{V}^{(i)} = \mathbb{R}^{A^{(i)} \times B^{(i)} \times \cdots \times C^{(i)}}$, and therefore layer outputs are referred to as **volumes**. The last tensor dimension of a volume is referred to as the **channel dimension**. [46, chapter 6]

The first layer is referred to as the **input layer** and its input domain is $\mathbb{X}$, i.e. $l^{(1)} : \mathbb{X} \mapsto \mathbb{V}^{(1)}$. The last layer $l^{(L)}$ is called the **output layer**. It might map directly onto the target domain, i.e. $\mathbb{V}^{(L)} = \mathbb{Y}$, or define an intermediate mapping such as probabilities that require subsequent binarization to map $\mathbb{V}^{(L)} \mapsto \mathbb{Y}$ (see section 2.2.3 on binarization).

All layers between the input and the output layer are referred to as **hidden layers**. [46, chapter 6]

The actual body of each function is left abstract in this explanation, as it is subject to the layer type. Modern neural networks usually mix various layer types.

Regardless of the layer type, the ith layer consists of a set of $U^{(i)} \in \mathbb{N}$ **units**. Units are functions that actually produce the values of the layer output volume. Often, but not necessarily, a layer has as many units as channels, i.e. $C^{(i)} = U^{(i)}$. The core properties of units are:

- All units in a layer have the same hypothesis space, i.e. they have the same function body but with possibly different weights. When we denote the parameter set of unit $u^{(i,j)}$ as $\boldsymbol{\theta}^{(i,j)}$, the set of all weights of a network f is defined as $\boldsymbol{\theta}_{\text{net}} = \{\boldsymbol{\theta}^{(1,1)}, \dots \boldsymbol{\theta}^{(L,U^{(L)})}\}$.
- Units process the layer input in parallel instead of sequentially (otherwise, they would be located in the next layer). [46, chapter 6]

The number of layers L and number of units U is also referred to as the **depth** and **width** of a model. However, there is no strict definition on how to count layers nor units. Often layers can be broken down into various sublayers. There is no consensus on how to count the width when layers have varying numbers of units inside the network. [46, chapter 6]

2.3.2 Selected layer types

In neural networks, layer types might be combined freely, with the only technical restriction being the requirements on each layers input domain. This sections presents a selected set of layer types $l^{(i)}$, which are relevant to the networks used in this thesis. However, for simplicity in reading, we omit the upper script layer index in this section to denote a layers simply as l.

2.3.2.1 Fully-connected layer

A **fully-connected layer** (**FCL**) (alias **dense layer**) has signature $l_{\text{dense}} : \mathbb{R}^N \mapsto \mathbb{R}^C$, i.e. it maps vectors of size N to vectors of size C. It contains a set of $U = C$ units, each one with signature $u^{(j)} : \mathbb{R}^N \mapsto \mathbb{R}$, where (j) is the unit index. In a FCL, a unit is also referred to as a **neuron**. A neuron is defined as

$$u(\boldsymbol{x}; \boldsymbol{\theta})^{(j)} = g(\langle \boldsymbol{w}^{(j)}, \boldsymbol{x} \rangle + b^{(j)}), \tag{2.27}$$

where $x \in \mathbb{R}^N$ is the input vector, $\boldsymbol{\theta} = \{\boldsymbol{w}, b, a\}$ is the set of unit parameters, $\boldsymbol{w} \in \mathbb{R}^N$ are the **weights**, $b \in \mathbb{R}$ is the **bias**, and $g : \mathbb{R} \mapsto \mathbb{R}$ is the **activation function**. A fully-connected layer defines a simple linear combination of the vectors $\boldsymbol{w}$ and $\boldsymbol{x}$, which are put through a non-linear transfer function g.

The jth component of the output vector is provided by the jth unit, i.e. when $l_{\text{dense}}(\boldsymbol{x}) = \boldsymbol{v}$ then $v_j = u^{(j)}(\boldsymbol{x})$, where the upper script (j) is the unit index. Consequently, the entire layer operation can be displayed as a matrix-vector product defined as

$$l_{\text{dense}}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}, g) = g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{2.28}$$

where g is the point-wise applied activation function, and $W \in \mathbb{R}^{C \times N}$ and $b \in \mathbb{R}^C$ are the collected weight matrix and bias vector from the units, so that $\boldsymbol{W}_{j,:} = \boldsymbol{w}^{(j)}$ and $\boldsymbol{b}_j = b^{(j)}$.

The activation function g is necessary, as stacking various fully-connected layers would otherwise collapse into a single linear combination.

The **sigmoid function** σ used to be the most common activation function for hidden layers in neural networks. It is defined as

$$\begin{aligned} \sigma &: \mathbb{R} \mapsto [0, 1] \\ \sigma(x) &= \frac{e^x}{1 + e^x}. \end{aligned} \tag{2.29}$$

However, it has fallen out of favor because of it attenuates the gradient flow through the network. The most common activation function for hidden layers in modern neural networks is the ReLU function [176, chapter 5] [46, chapter 6]

$$ReLU(x) = \max(0, x). \tag{2.30}$$

The central hyperparameter of a FCL is the activation function and the amount of units.

Comparing this definition with logistic regression presented in section 2.2.5.1, it is obvious that logistic regression is a special case of a fully-connected layer.

2.3.2.2 2D Convolutional layers

Units in convolutional layers perform the convolution operation. In this context, units are also referred to as **filters**. There are different kinds of convolutional layers. For this thesis, only **2D convolutional layers** are of interest.

A 2D convolutional layer has signature $l_{\text{2D-conv}}(\boldsymbol{X}) = \boldsymbol{V}$, where $\boldsymbol{X} \in \mathbb{R}^{W \times H \times C'}$ and $\boldsymbol{V} \in \mathbb{R}^{W \times H \times C}$. The volume dimensions W, H and C are interpreted as the volume width, height, and channels, respectively. Filters produce $u^{(j)}(\boldsymbol{X}) = \boldsymbol{O}^{(j)}$, where $\boldsymbol{O} \in \mathbb{R}^{W \times H}$, so that $\boldsymbol{V}_{:,:,j} = \boldsymbol{O}^{(j)}$. The convolution operation of a filter is point-wise defined as

$$O_{w,h} = (\boldsymbol{X} \star \boldsymbol{K})(w, h) = g(b + \sum_{m=1}^{W_{\text{kernel}}} \sum_{n=1}^{H_{\text{kernel}}} \langle X_{w-m,h-n,:} \ , \ K_{m,n,:} \rangle), \tag{2.31}$$

where $w \in \{1, \ldots, W\}$ is the width index, $h \in \{1, \ldots, H\}$ is the height index, $\boldsymbol{K} \in \mathbb{R}^{W_{\text{kernel}} \times H_{\text{kernel}} \times C}$ is the **filter kernel**, $b \in \mathbb{R}$ is the bias, and g is an activation function as in a FCL.

A filter calculates a linear combination of its input followed by an activation function, similar to a FCL. However, contrary to a FCL, it only considers a limited range of input values in width and height according to the filter kernel size W_{kernel} and H_{kernel} at each position. Consequently, the filter kernel size is also referrred to as the **receptive field**. A unit in a convolutional layer is specialized in finding a specific pattern, associated with its filter kernel, at different spatial locations w, h of the input volume. [176, chapter 5][46, chapter 9]

Usually, input volumes are zero-padded at the borders so that the input and output volume have the same dimension sizes for width and height. This is referred to as **same padding**. If there is no padding, those dimensions shrink accordingly.

A variation to these layers are **strided convolutional layers**, which define a **stride** $S \in \mathbb{N}$, which determines the spacing between points at which convolutional results are calculated $w \in \{S \cdot 1, S \cdot 2, \ldots, W\}$ and $h \in \{S \cdot 1, S \cdot 2, \ldots, H\}$. [176, chapter 5][46, chapter 9]

The central hyperparameter of a convolutional layer is the activation function, the kernel size (widht and height), the stride, and the amount of filters.

2.3.2.3 2D Pooling layers

2D pooling layers operate on three dimensional input volumes $\mathbb{R}^{W \times H \times C}$ and are usually combined with 2D convolutional layers. They reduce the input dimensionality along the width W and height H dimension. They operate similar to convolutional layers. However, instead of performing convolutional operations on the input receptive field, they perform a parameter-free statistical summary of the input receptive field.

The layer operation is point-wise defined as

$$l_{\text{pool}}(\boldsymbol{X})_{w,h,c} = p(\boldsymbol{X}_{w:w+n,h:h+m,c}), \tag{2.32}$$

where $n \in \mathbb{N}$ and $m \in \mathbb{N}$ are the width and height of the pooling kernel, and $p \in \mathbb{R}^{n \times m} \mapsto \mathbb{R}$ is some statistical function, such as maximum or average.

As convolutional layers, padding of input volume is common. To achieve the reduction in volume size, we stride the pooling layer. [46, chapter 9][176, chapter 5]

The most common pooling operation in CNNs currently is max pooling, i.e. calculation of the maximum.

The main hyperparameters of a pooling layer are the kernel size, stride, and pooling operation.

2.3.2.4 Recurrent layers

Recurrent layers are specialized in processing time series with T time steps. The input volume is $\boldsymbol{X} \in \mathbb{R}^{T \times C}$, where C is the channel axis. Consequently, $\boldsymbol{X}_{t,:}$ are the features at the tth time step. The output volume is $\boldsymbol{V} \in \mathbb{R}^{T \times C'}$. To produce the output, the layer applies the same weight vector $\boldsymbol{w} \in \mathbb{R}^{C'}$ at each time step $\boldsymbol{X}_{t,:}$ similar to a FCL. Additionally, the layer incorporates the output at the previous time step $\boldsymbol{V}_{t-1,:}$ in its

own layer through another weight vector $\boldsymbol{r} \in \mathbb{R}^{C'}$. Consequently, we calculate the output at time step t as

$$\boldsymbol{V}_{t,:} = g(\langle \boldsymbol{w}, \boldsymbol{X}_{t,:} \rangle + \langle \boldsymbol{r}, \boldsymbol{V}_{t-1,:} \rangle + b), \tag{2.33}$$

where g is an activation function and $b \in \mathbb{R}$ is the bias. Because of the recurrent calculation procedure, the output at step t actually involves all previous states $1, \ldots, t-1$. [176, chapter 5]

When we forward the entire output volume $\boldsymbol{V}$, this is referred to as as *Many-to-many* translation. However, it is also common to just forward the last output at $\boldsymbol{V}_{T,:}$, which is referred to as *many-to-one* translation. The first case is applied when we aim to stack various recurrent layers. The latter case is common when we aim to collapse the time dimension, e.g. when classifying an entire time series. [176, chapter 5]

Training of this layer requires *back-propagation through time*, where we unroll all time steps to approximate the gradient. However, in practice this leads to vanishing and exploding gradients, due to cumulative effects. [176, chapter 5]

In practice, we use more sophisticated extensions of recurrent layers, which extend the naive implementation to combat the vanishing and exploding gradient problem. The two most common types are **long short-term memorys** (**LSTMs**) [62] and **gated recurrent units** (**GRUs**) [27]. The latter was proposed as a simplification of the former. Both define additional *gates* with weight vectors, which control the internal state to regulate the information flow through time, e.g. there are gates for gradual "forgetting" of information. Stating the specifics of these implementations is beyond the scope of this thesis. I recommend interested readers the original publications. [176, chapter 5]

2.3.2.5 Batch normalization

Batch normalization was proposed by Ioffe and Szegedy [68]. Batch normalization normalize the values of the input volume along the indicated axis through rescaling and recentering. For many networks, batch normalization significantly improves training time and convergence of the network.

Let $\boldsymbol{x} \in \mathbb{R}^N$ be an input vector. At the forward pass, we normalize each value as

$$l(x_i) = \left(\frac{x_i - \mu_i}{\sqrt{\sigma_i^2}}\right) \cdot \alpha_i + \beta_i, \tag{2.34}$$

where σ_i^2 and μ_i are the mean and standard deviation of the input calculated on the input training batch and α_i and β_i are learnable parameters that might offset the normalization (see section 2.3.3.2 on the term *batch*).

Although this example is aimed at vector inputs, this approach is applicable to input volumes of arbitrary dimensionality by specifying the axes over which to calculate statistics accordingly. This layer is usually applied before the activation function of a convolutional or fully-connected layer.

To apply batch normalization at test time to singular examples, we keep a running average of the training mean and standard deviation, where a *momentum* parameter governs the influence of the latest batch on this average.

2.3.2.6 Other layers

This section presents a list of other layers with importance to this thesis:

- **Flatten layer**: This layer flattens the input volume to a vector.
- **Time-distributed layer**: This layer wraps another layer. It applies the same layer at each time step of an input volume
- **Dropout**: This layer randomly sets a certain percentage of input activations to zero at each training epoch. Consequently, it induces noise to the activations at training time, which can reduce overfitting in neural networks. It was proposed by Srivastava et al. [162].
- **Global pooling layers**: This layer is similar to the already introduced 2D pooling layer. However, it performs the pooling operation across the input spatial dimensions. A 2D global pooling inputs 3D input volumes and pools across the entire width and height. A 1D pooling operation inputs a 2D input volume and pools across the first dimension. The most common variants of this are **global average pooling** (**GAP**) and **global max pooling** (**GMP**), which pool the average or the maximum across the input volume dimensions, respectively.

2.3.3 Training of neural networks

As described in section 2.2.1, optimization of the model weights requires various components, such as a loss function and an optimization algorithm. This section defines those components for neural networks.

2.3.3.1 Loss calculation

As stated in section 2.2.1, the loss of an example is calculated through a (surrogate) loss $L_{\text{loss}}(\boldsymbol{y}, \hat{\boldsymbol{y}})$ (in this section I highlight the loss through the subscript text to avoid confusion with the number of layers L). As the model $f(x)$ is a combination of functions $l^{(L)}(\dots l^{(1)}(\boldsymbol{x}))$, the actual prediction is produced by the last layer $\hat{\boldsymbol{y}} = l^{(L)}(\dots)$.

Consequently, we must determine three components for loss calculation: The encoding of the target $\boldsymbol{y}$, the content of the last layer $l^{(L)}$, and a surrogate loss function L_{loss}. All components must be chosen in combination with each other to fit the task. While theoretically these components can be chosen arbitrarily, there are certain established default choices for certain tasks. [46, chapter 6][176, chapter 5]

I limit the loss calculation to the tasks binary, multi-class, and multi-label classification (see section 2.2.3), as only those were of interest to this thesis' studies. For these tasks, neural networks employ FCLs as output layers. They use activation functions that limit the output layer codomain to the range $[0, 1]$. Consequently, neural networks

output class probabilities $\hat{\boldsymbol{p}}$, i.e. these probabilities must be subsequently binarized (see section 2.2.3 on binarization). [46, chapter 6][176, chapter 5]

Binary classification tasks use integer encoded targets with a single output neuron. The activation function is the sigmoid function (equation 2.29). The loss is **binary cross entropy** (**BCE**), which is defined as

$$L_{\text{loss}}(y, \hat{p}) = \begin{cases} -\log(\hat{p}) & \text{, if } y = 1 \\ -\log(1 - \hat{p}) & \text{, else.} \end{cases} \tag{2.35}$$

Multi-class and multi-label classification tasks use one-hot encoded target vectors with as many output neurons as classes. In multi-class classification, the output activation function is softmax (equation 2.26). Consequently, the output layer corresponds to the definition of the multinominal logistic regression as defined in section 2.2.5.1. The loss is **categorical cross entropy** (**CCE**), which is defined as:

$$L_{\text{loss}}(\boldsymbol{y}, \hat{\boldsymbol{p}}) = -\sum_i y_i \cdot log(\hat{p}_i) \tag{2.36}$$

In multi-label classification, the output activation function is sigmoid. The loss is multi-label BCE, which is defined as [46, chapter 6]

$$L_{\text{loss}}(\boldsymbol{y}, \hat{\boldsymbol{p}}) = \sum_i -y_i \log(\hat{p}_i) - (1 - y_i) \log(1 - \hat{y}_i). \tag{2.37}$$

All of these losses weight all classes equally. However, as discussed in section 2.2.4.1, settings with class imbalance cause the majority classes to dominate the average loss. To compensate for this imbalance, there are alternative definitions of the above mentioned losses. One of the most prevalent variant is associating classes with fixed weights [72, 77, 94]. **Weighted BCE** is defined as

$$L_{\text{loss}}(y, \hat{p}) = \begin{cases} -w_1 \log(\hat{p}) & \text{, if } y = 1 \\ -w_0 \log(1 - \hat{p}) & \text{, else,} \end{cases} \tag{2.38}$$

and **weighted CCE** is defined as

$$L_{\text{loss}}(\boldsymbol{y}, \hat{\boldsymbol{p}}) = -\sum_i w_i \cdot y_i \cdot log(\hat{p}_i), \tag{2.39}$$

where $w_i \in \mathbb{R}$ is the class weight associated with class i. King and Zeng [77] proposed to choose weights according to the relative count of examples, so that $w_i = |C_i|/(|K| \cdot |C|)$, where $i \in K = \{0, 1\}$ is the class index, C_i is the set of examples for class i, and C is the total set of examples.

2.3.3.2 Optimization through gradient descent

Training of a neural network means adapting its weights $\boldsymbol{\theta}_{\text{net}}$ to optimize the loss on the training set. The total loss is calculated as the empirical risk as defined in equation

2.11, using one of the previously shown loss functions. However, there is no analytical solution to equation 2.11. Alternatively, we use an iterative method of optimization by randomly initializing model weights and gradually adapting them to iteratively decrease the loss, which is known as **gradient descent**.

For this, one calculates the **gradient** of the loss with respect to the model weights:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \mathbb{S}) = \frac{1}{|\mathbb{S}|} \nabla_{\boldsymbol{\theta}} \sum_{\boldsymbol{x},\boldsymbol{y} \in \mathbb{S}} L_{\text{loss}}(\boldsymbol{y}, f(\boldsymbol{x})), \tag{2.40}$$

where variables correspond to the definitions in section 2.2.1.

Calculating the gradient is not straight forward, as f is a composite function. The gradient must be calculated with respect to every weight in every layer. This requires repeated application of the chain rule, which is referred to as **back propagation**. The details of this procedure are beyond the scope of this thesis. Interested readers are recommended to read Goodfellow et al. [46, chapter 8].

We update the weights according to the gradient. A **weight update** is defined as

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \mathbb{S}), \tag{2.41}$$

where $\eta \in \mathbb{R}$ is the **learning rate**.

Stochastic Gradient Descent (SGD) is an algorithm for training a neural network based on these principles: In a loop, we repeatedly calculate the loss on the entire training set $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \mathbb{S} = \mathbb{S}_{\text{train}})$ according to equation 2.11 and update the weights according to equation 2.40. Each pass through the training set is referred to as an **epoch**.

However, computing the gradient over a large data set is computationally expensive. In practice, we use **Mini batch stochastic gradient descent** instead. The training set is subdivided into disjoint subsets $\mathbb{B}^{(1)}_{\text{train}}, \mathbb{B}^{(2)}_{\text{train}}, \ldots$ of equal size, where a subset is referred to as a **batch**. The **batch size** $B = |\mathbb{B}_{\text{train}}|$ is the amount of examples in a batch. Weight updates are then calculated according to the gradient of the batch $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \mathbb{S} = \mathbb{B}_{\text{train}})$. An **epoch** is defined as an pass through the entire training set, i.e. there are $\mathbb{S}_{\text{train}}/B$ weight updates per epoch. Mini batch gradient descent is so common that it is synonymous to gradient descent, i.e. SGD usually means mini batch gradient descent. [46, chapter 8]

Optimization Algorithms: There are numerous optimization algorithms that offer more sophisticated approaches for calculation of the gradient and applying the weight update. For example, we might want to adapt the learning rate throughout training or include second-order derivatives into the gradient. Optimization algorithms for neural networks are an ongoing field of research. An exhaustive presentation of these algorithms is beyond the scope of this thesis. The optimization algorithm that currently is most common in the ASR community is **ADAM**, presented by Kingma and Ba [78]. [46, chapter 8]

Early stopping: The number of epochs is a hyperparameter. It might be set fixed prior to starting the training. However, it can also be chosen dynamically during training: Figure 2.6 visualizes the typical development of the training and validation/test

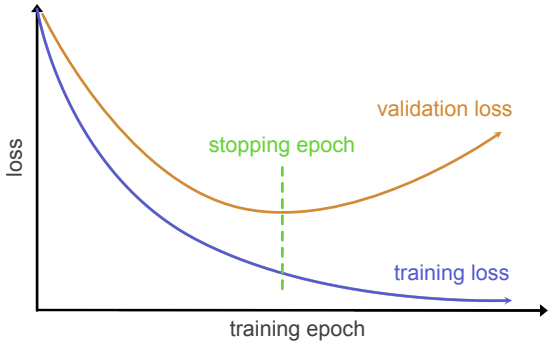

Figure 2.6: **Schematic visualization of early stopping in neural network training**

set loss across epochs. Weights of neural networks are initialized randomly following a predetermined initialization theme. At the start of training, neural networks display an at-chance performance. During training, the training loss decreases monotonically. However, the validation/test loss usually decreases first and later increase again. This is due to the high capacity of neural networks, causing them to overfit as training progresses. Therefore, we use the validation set to monitor the loss at each epoch and stop training at the optimal epoch. This process is referred to as early stopping. The primary hyperparameter is *patience*, which is the number of epochs we wait for the loss to improve further before stopping training and returning to the best epoch. [46, chapter 7]

2.3.4 Common architectures

The architecture of a neural network is its layer composition and configuration. Architecture design offers immense degrees of freedom: We can select from a wide range of layer types, of which only a subset was displayed in section 2.3.2. All layers can be combined freely, stacking layers of the same or different types. The only restriction to combining layers are the layers' input and output volume demands. Furthermore, all layers carry various hyperparameters, such as amount of units. Consequently, there is an infinite number of possible architectures. The big challenge in neural networks is that the entire network is a hyperparameter.

Designing of performant architectures is more of an art than a science. There is no analytical method to calculate the perfect architecture for a given task deterministically, just based on knowledge of the task. Architectures are usually developed by the researcher making assumptions about the nature of the task and designing an architecture accordingly. Automatic search of architectures is an ongoing field of research [167].

In this section I present some common architecture templates that currently are prevalent for a variety of tasks, particularly in ASR. All of these architectures were of particular importance to studies (B) – (D). Figure 2.7 visualizes these networks.

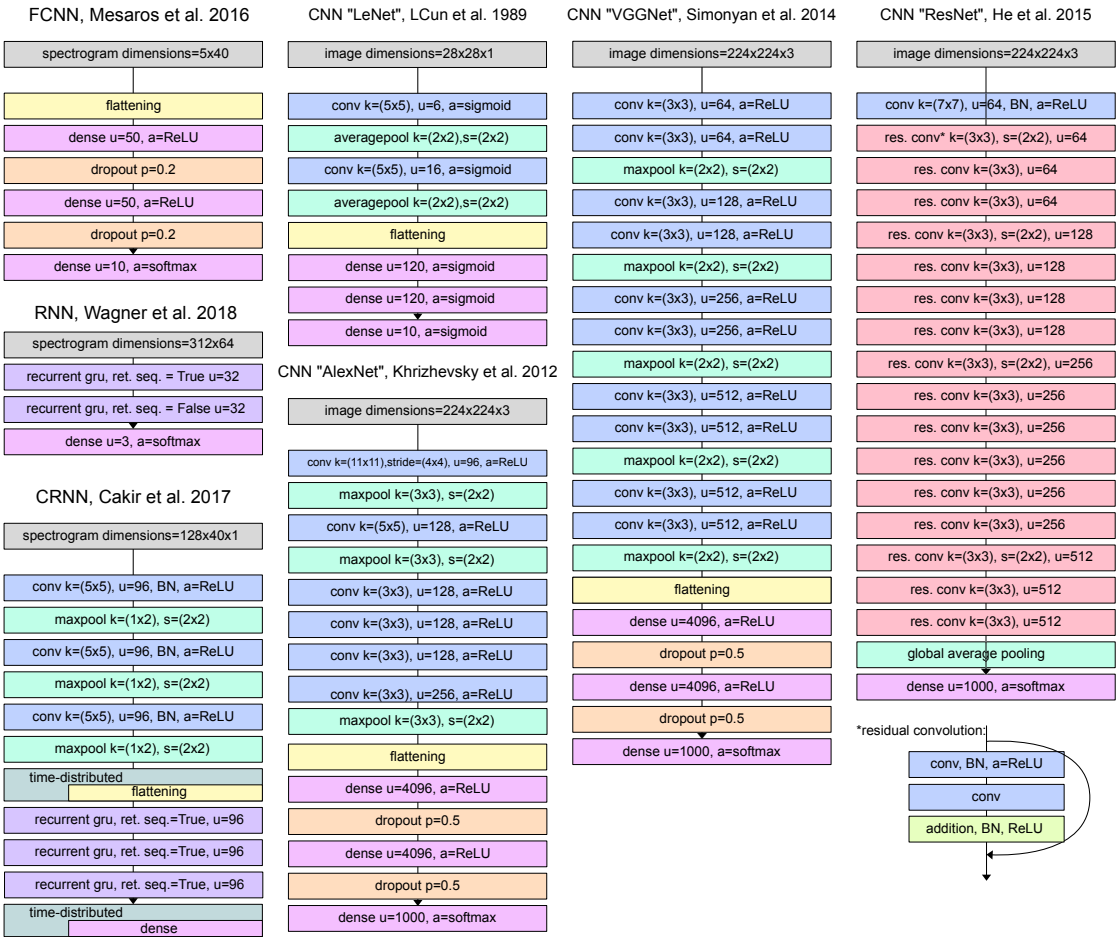

Figure 2.7: **Overview over selected network architectures.** The left column contains three networks for audio recognition, the right three columns contains networks for computer vision. Colors encode layer types. Each layer states its parametrization. Abbreviations: dense = fully connected layer; conv = 2D convolutional layer; maxpool = 2D max pooling; res. conv. = residual convolutional layer; u = number of units; a = activation function; k= kernel size; s = stride; BN = batch normalization; ret. seq. = return sequences. Sources of networks: [22, 55, 85, 90, 106, 155, 177]

2.3.4.1 Fully-connected neural networks

Fully connected neural networks (FCNNs) are neural networks that consist primarily off fully connected layers. The central design choice in these networks is the amount of layers and amount of units in each layer. In the literature, these networks are also referred to as multi layer perceptrons (MLPs). [176, chapter 5]

Technically, fully connected layers require their input volume to be a vector. Otherwise, the input needs to transformed accordingly, e.g. through a flattening layer. A FCL and network assumes that the order of input vector components conveys no mean-

ing, i.e. neighboring components in a vector are not assumed to be more similar than components that are far apart.

Figure 2.7 shows an exemplar FCNN. The network was presented by Mesaros et al. [106] for the DCASE 2016 task 1 baseline system. The network inputs a 2D feature map and therefore requires a flattening layer to input it to the network.

2.3.4.2 Recurrent neural networks

Recurrent neural networks (RNNs) are neural networks that consist primarily of recurrent layers. The central design choice in these networks is the choice of the recurrent layer type, the amount of layers and amount of units per layer.

Figure 2.7 shows an exemplar recurrent network, consisting of two recurrent layers. The network was used by Wagner et al. [177] for the 2018 ComParE challenge. The network has two recurrent layers, where the first one returns the output at each time step and the second one returns only the last state. The RNN requires choosing which of the input dimensions is considered the time axis.

Recurrent networks assume that the index in the time step axis carries meaning, i.e. that neighboring time steps are more similar than time steps that are further away. The index of the feature axis carries no meaning. [176, chapter 5]

2.3.4.3 Convolutional neural networks

Convolutional neural networks (CNNs) are neural networks that contain convolutional layers as the primary building blocks in the architecture. However, they contain other layers as well, most importantly pooling layers, and sometimes FCLs. The right three columns of Fig. 2.7 show a selection of exemplar CNNs that had a particularly high impact on the research community. All of these architectures were developed for computer vision.

The network *LeNet* by LeCun et al. [90] is often referred to as *the* first CNN. It was proposed for the task of recognition of hand-written digits. It introduced various design principles, such as: alternating convolutional layers with pooling layers; reducing volume size with each pooling layer through striding; using quadratic kernels for convolutional and pooling layers; following convolutional layers with a stack of fully-connected layers; using flattening to adapt the convolutional layer output to the fully-connected layers. [46, chapter 9]

AlexNet was proposed by Krizhevsky et al. [85] for the ImageNet competition in 2012. The key differences to LeNet are: it is deeper; it uses ReLU activation function, which proved to outperform sigmoid activation used in LeNet; it uses dropout in between fully-connected layers to improve generalization [162]. AlexNet gained popularity, as it started the "deep learning era" due to it significant improvement on the competing systems in the ImageNet challenge that were not neural network based. Following AlexNet, most modern image recognition solutions are based on CNNs. Up until today, it is not clear why CNN were "forgotten" after LeNet for over 20 years. [46, chapter 9]

VGGNet was proposed by Simonyan and Zisserman [155] for the ImageNet competition in 2014 and improved on the performance of AlexNet. It gained popularity for its simplistic architecture and set implicit standards for coming designs in CNNs. The key differences to AlexNet and LeNet were: greater depth; the usage of smaller 3×3 kernel sizes for convolutional layers and 2×2 kernel sizes for pooling layers; doubling the amount of convolutional kernels after each pooling operation; keeping kernel sizes identical across the entire network.

ResNet was proposed by He et al. [55] for the ImageNet competition in 2015 and improved on the performance of VGGNet. It is arguably the default pretrained network for any computer vision task to date. It introduced *residual convolutional layers*, which is a stack of two convolutional layers with a skip connection. According to He et al. [55], these allow better training in very deep networks. Additionally, it popularized various design principles, such as: Greater depth; following convolutional layers with batch normalization [68]; using global pooling instead of flattening for adapting volume dimensionality after the convolutional stage; dismissing the FCLs completely, apart from the output layer.

Consequently, there was an ongoing trend to increasing depth in networks, with smaller kernel sizes. The usage of batch normalization has become quasi-standard in CNNs and has been added to more modern adaptions of VGGNet [32, 60]. Even though all of these CNN examples were developed for image recognition, they have been used for audio recognition as well with success [60]. Section 3.2 elaborates further on this.

2.3.4.4 Convolutional recurrent neural networks

Convolutional recurrent neural networks (CRNNs) are networks that combine convolutional layers with recurrent layers. These networks originate from speech recognition [143] and have been popularized by Cakır et al. [22] for audio detection tasks. Fig. 2.7 shows one of those networks from Cakır et al. [22]. It outputs predictions at each time step and therefore uses a time-distributed fully-connected layer at the output. These networks do not reduce volume dimensionality across the time-axis to retain alignment between the input feature map and output targets. Therefore, the volume dimensionality adaption between convolutional and recurrent layers is implemented through a time-distributed operation as well.

2.4 Automatic sound recognition through machine learning

Automatic sound recognition (ASR) is the umbrella term for computational tasks with the goal of automatic, computational prediction of information about an input audio signal. ASR is primarily concerned with sound classification and detection. This comprises all kinds of sounds: every-day sound classes such as a siren, but also non-verbal communication events specifically. [176, chapter 1]

ASR is not a mere subdisicipline of automatic speech recognition. Both ASR and automatic speech recognition operate on audio signals, which for some ASR tasks can involve the human voice. However, the output domain of speech recognition is much more complex: While ASR usually only outputs classes of sound events with or without time frames of occurrence, automatic speech recognition outputs text of the spoken speech. The latter is much more complex than the former, e.g. same-sounding vocalizations can mean different written words depending on the context. While there is some overlap in the methods of both research disciplines, ASR is better understood as its own research discipline, rather than as a derivative of automatic speech recognition. Also, methods used in ASR overlap just as much with image recognition as they do with automatic speech recognition. [176, chapter 1]

The goal of this section is to introduce the basics of ASR. This section is in close relationship with all previous foundations sections: Section 2.1 introduced the type of acoustic signals and event classes of interest. Section 2.2 introduced ML, which is the currently leading approach for developing ASR systems. Section 2.3 introduced neural networks, which currently are the most common type of ML model in deep learning based ASR systems.

The structure of this section is as follows. Section 2.4.1 specifies the core ASR tasks. Section 2.4.2 presents audio representations. Section 2.4.3 discusses methods for prediction of sound information, based on these sound representations. Finally, section 2.4.4 highlights particularities in evaluation of ASR systems.

2.4.1 Overview of ASR tasks

While in section 2.1.1 we defined audio signals as functions $x : \mathbb{N} \mapsto \mathbb{R}$, here we define them as vectors $\boldsymbol{x} \in \mathbb{R}^{L_{\text{sig}}}$ of length L_{sig}. Let f be the function performing the predictive task on $\boldsymbol{x}$.

ASR fundamentally knows two types of tasks, which are visualized in Fig. 2.8:

- **Sound classification / regression** is the prediction of information about an audio signal *without* temporal allocation as $f(\boldsymbol{x}) = \hat{\boldsymbol{y}}$, where $\hat{\boldsymbol{y}} \in \mathbb{Y}$ is the output information. This corresponds to conventional classification or regression as defined in section 2.2.3, where the input is an audio signal. In ASR, *monophone* classification is synonymous to binary or multi-class classification, and *polyphone* to multi-label classification.
- **Sound detection** is the prediction of information about an audio signal *with* temporal allocation. The analysis function is defined as

$$f(\boldsymbol{x}) = \{(\hat{\boldsymbol{y}}^{(1)}, t^{(1)}_{\text{start}}, t^{(1)}_{\text{stop}}), \ldots, (\hat{\boldsymbol{y}}^{(O)}, t^{(O)}_{\text{start}}, t^{(O)}_{\text{stop}})\}, \tag{2.42}$$

  where $\hat{\boldsymbol{y}}^{(i)} \in \mathbb{Y}$, $t^{(i)}_{\text{start}}$ and $t^{(i)}_{\text{stop}}$ are the target, start and stop time of of event i. As in audio classification / regression, the domain of y might be nominal or continuous. There are two major types of detection tasks: (1) monophone detection, where only one sound event can be active at a time, and (2) polyphone detection, where any number of events can be active at a same time. [176, chapter 2]

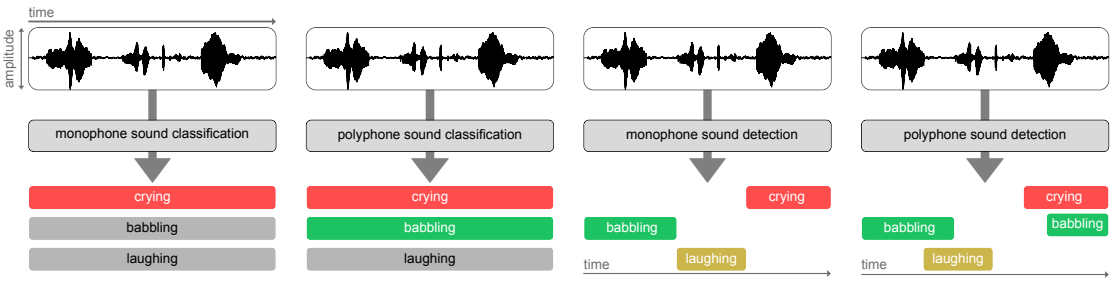

Figure 2.8: **Schematic visualization of the major audio recognition tasks by an example.** The top signal is the input waveform $\boldsymbol{x}$. The bottom shows the activity of three target classes $y \in \{\text{crying, babbling, laughing}\}$. As shown, in this case only polyphone systems are able to infer absence of a target class. Figure inspired by [176, chapter 2].

Audio detection departs from conventional classification and regression, since the output domain is more complex. However, detection can be solved through sound classification. For this, we might segment the input signal into a list of shorter segments and classify these segments separately. The segment duration and overlap determines the temporal resolution of target activity indications. Consequently, any system originally trained for audio classification might be transferred to detection. [176, chapter 2]

2.4.2 Audio representation and features

As introduced in section 2.2.1, machine learning requires examples to be represented as features. In theory, it is possible to consider each signal sample x_i directly as a feature. However, signals are usually represented as *feature maps* $\boldsymbol{M} \in \mathbb{R}^{T \times F}$, i.e. a two dimensional matrix , where $T \in \mathbb{N}$ is the number of time steps and $F \in \mathbb{N}$ is the number of features.

The general approach for transforming a signal into a feature map is as follows, which is visualized in Fig. 2.9:

(1) Framing: The signal is segmented into a series of **frames** or **windows**. This process is analogous to the framing performed for calculation of the STFT presented in section 2.1.1.3, however without the direct frequency transformation. As in the STFT, this framing process is specified by the frame length L_{frame} and the hop size H between frames. Typical frame lengths are 20 – 40 ms and typical hop sizes are 25 % – 100 % of the frame length. The framing process converts $\boldsymbol{x} \in \mathbb{R}^{L_{\text{sig}}}$ into a series of frames $\{\dot{\boldsymbol{x}}^{(1)}, \ldots, \dot{\boldsymbol{x}}^{(T)}\}$, where each element is defined as

$$\dot{\boldsymbol{x}}^{(t)} = [\boldsymbol{x}_{H \cdot t}\ , \ldots,\ \boldsymbol{x}_{H \cdot t + L_{\text{frame}}}]^{\top}, \tag{2.43}$$

where $t \in \{1, \ldots, T\}$ is the time frame index, with $T = \lfloor \frac{L_{\text{sig}}}{H} \rfloor$.

(2) Extraction of time domain features: We extract a set of features on each time domain frame $\dot{\boldsymbol{x}}^{(t)}$. Each feature is extracted through a function $f_{\text{t-feat}}(\dot{\boldsymbol{x}}^{(t)}) = v$, where each feature $v \in \mathbb{R}$ represents some kind of summative property of the frame. This

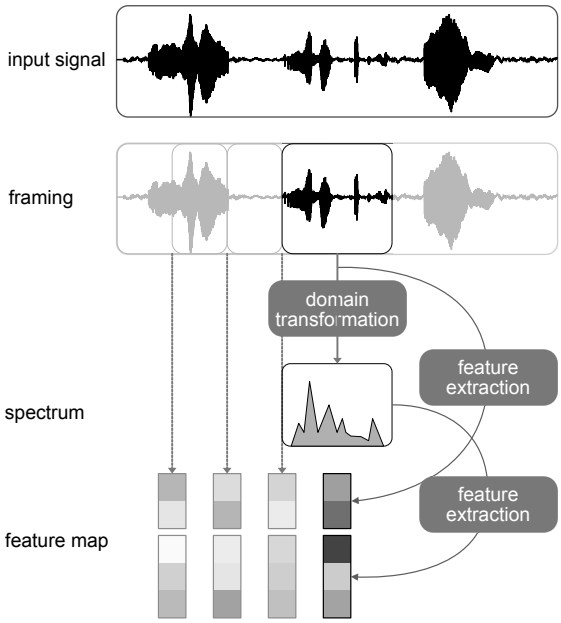

Figure 2.9: **Schematic visualization of the sound feature extraction process.** Figure inspired by [176, chapter 2].

way, we receive a feature vector for each time frame as $\boldsymbol{v}^{(t)}_{\text{t-feat}}$ with as many features as extraction functions. Typical features are: average energy, zero-crossing rate, or auto correlation coefficients.

(3) Frequency domain transformation: Each time domain frame $\dot{\boldsymbol{x}}^{(t)}$ is transformed into the frequency domain through the DFT defined in equation 2.1, so that $DFT\{\dot{\boldsymbol{x}}^{(t)}\} = \dot{\boldsymbol{s}}^{(t)}$. We might apply a window function such as the hanning window as defined in equation 2.5. We receive a sequence of spectral frames $\dot{\boldsymbol{s}}^{(t)}, \ldots, \dot{\boldsymbol{s}}^{(T)}$.

(4) Extraction of frequency domain features: Analogous to step (2), we extract a set of features based on each spectral frame, where each feature represents some type of summative property on the frame's spectral characteristic. The resulting spectral feature vector is $\boldsymbol{v}^{(t)}_{\text{f-feat}}$.

(5) Collection into feature matrix: We collect all features into a feautre matrix $\boldsymbol{M}$:

$$\boldsymbol{M} = \begin{bmatrix} {}_{\text{t-feat}}v_1^{(1)} & \cdots & {}_{\text{t-feat}}v_1^{(T)} \\ {}_{\text{t-feat}}v_1^{(2)} & \cdots & {}_{\text{t-feat}}v_2^{(T)} \\ \cdots & \cdots & \cdots \\ {}_{\text{f-feat}}v_1^{(1)} & \cdots & {}_{\text{f-feat}}v_1^{(T)} \\ {}_{\text{f-feat}}v_2^{(2)} & \cdots & {}_{\text{f-feat}}v_2^{(T)} \\ \cdots & \cdots & \cdots \end{bmatrix} \tag{2.44}$$

We might insert an arbitrary number of additional transformations-and-feature-extraction-chains before collecting the results into the final feature map. For example, some applications transform the spectral domain into the cepstrum domain to extract so-called cepstral features.

The simple spectrogram representations presented in section 2.1.1.3 are special cases of this approach, where the features directly correspond to the intermediate frequency representation. Consequently, any spectrogram visualization might be used as a feature map $\boldsymbol{M}$.

The key principle in this approach is that we calculate features on short frames whose content we consider quasi stationary. However, as feature extraction functions are applied equally at each time frame, they are agnostic to the location or duration of frames. In this context, features are also referred to as **low level descriptors** (**LLD**s) in the community of computational paralinguistics. Figure 2.9 visualizes this feature extraction process. [176, chapter 4][148, chapter 8]

2.4.3 Sound prediction through machine learning models

This section presents a generalized pipeline for solving audio recognition tasks through ML models. I designed this framework as a general template into which various approaches discussed in this thesis can be placed. It outlines the most essential steps, while actual pipelines are often more complex. This framework is based on the pipelines presented by Hershey et al. [60], Schuller and Batliner [148], Virtanen et al. [176].

Let $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$ be a a set of N audio signals in the time domain. We perform classification or detection on each of these signals as follows. Fig. 2.10 visualizes the process.

1. **Time domain preprocessing**: Preprocessing has the goal of manipulating signal properties to facilitate the prediction process. Common preprocessing steps are mono-conversion of stereo signals or amplitude normalization.
2. **Feature map conversion**: We convert signals $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$ to feature maps $\{\boldsymbol{M}^{(1)}, \ldots, \boldsymbol{M}^{(N)}\}$ according to the process outlined in section 2.4.2. Each feature map $\boldsymbol{M}^{(i)} \in \mathbb{R}^{T^{(i)} \times F}$ has the same number of features F, but a different times axis length $T^{(i)}$ according to the duration of the respective signal.
3. **Segmentation**: The goal of segmentation is to bring all feature maps to the same temporal duration $T_{\text{seg}} \in \mathbb{R}$. This step is only necessary if the model requires equally-sized feature maps for training and testing. If $T_{\text{seg}} < T^{(i)}$, we need to segment it into segments of fixed length T_{seg} using some hop size $H_{\text{seg}} \in \mathbb{R}$. Consequently, an input feature map $\boldsymbol{M}^{(i)} \in \mathbb{R}^{T^{(i)} \times F}$ is converted into a list $\{\dot{\boldsymbol{M}}^{(i,s)}\}_{s=1,\ldots,S^{(i)}}$, with $\dot{\boldsymbol{M}}^{(i,s)} \in \mathbb{R}^{T_{\text{seg}} \times F}$, where s is the segment index and $S^{(i)} = \lfloor T^{(i)} / H_{\text{seg}} \rfloor$ is the amount of segments. This segmentation is similar to the framing process for feature extraction presented in section 2.4.2, just that the segment and hop size usually span various seconds, instead of milliseconds. If

$T_{\text{seg}} > T^{(i)}$, we need to pad or loop signal (i) to bring its length up accordingly. Let e be such a padding function, then $\dot{\boldsymbol{M}}^{(i,1)} = e(\boldsymbol{M}^{(i)})$.

4. **Temporal feature aggregation**: Some models are unable to directly process two-dimensional matrices, for example conventional machine learning models (see section 2.2.5) or pure fully-connected neural networks (see section 2.3.4). In this case, we need to remove the temporal axis of the feature map segments through some form of aggregation function $agg(\dot{\boldsymbol{M}}^{(i,s)}) = \dot{\boldsymbol{m}}^{(i,s)}$, where $\dot{\boldsymbol{m}}^{(i,s)} \in \mathbb{R}^{F'}$ is the resulting feature vector. We achieve this through feature-wise statistical summaries across time, such as calculation of the mean, max, standard deviation etc. We might also simply flatten the feature map, however this requires the prior segmentation step if the model requires same-sized inputs.
5. **Prediction**: The model predicts each segment individually, which is either $f(\dot{\boldsymbol{M}}^{(i,s)}) = \hat{\boldsymbol{y}}^{(i,s)}$ or $f(\dot{\boldsymbol{m}}^{(i,s)}) = \hat{\boldsymbol{y}}^{(i,s)}$. As discussed in section 2.2.3, some models output class probabilities instead of directly mapping onto the target domain, i.e. they output $\hat{\boldsymbol{p}}^{(i,s)}$.
6. **Output aggregation**: If the recognition task is a detection task, the segment-wise probabilities $\{\hat{\boldsymbol{y}}^{(i,1)}, \ldots, \hat{\boldsymbol{y}}^{(i,S^{(i)})}\}$ directly indicate temporal allocations of predictions. For classification tasks, we need to aggregate the segment-wise outputs through a function $agg(\{\hat{\boldsymbol{y}}^{(i,1)}, \ldots, \hat{\boldsymbol{y}}^{(i,S^{(i)})}\}) = \hat{\boldsymbol{y}}^{(i)}$, similar to the temporal aggregation of features. One common aggregation function is majority voting, i.e. setting the most common voted label in a classification setting.
7. **Output binarization**: If the task is classification and the model outputs probability scores, we need to binarize outputs as discussed in section 2.2.3. Some pipelines perform binarization on segments before aggregation.

Most of these steps are optional: The prediction step (5) is the only strictly required one. Particularly steps (3) and (4) have similar goals: Both aim at eliminating the variation of the time axis length across different feature maps. Therefore, both or either one of them might be omitted if the model is able to handle feature maps of different sizes for training and testing.

Choosing the segment length T_{seg} in the segmentation step is a non-trivial hyperparameter. It determines the amount of context involved in prediction. It represents a tradeoff between computation time on one hand and retaining sufficient context to allow class separability on the other hand. Choosing T_{seg} too small might lead to excessive truncation of longer segments, which might dismiss information important for class separability; choosing T_{seg} too large might lead to extensive padding of shorter segments which needlessly increases processing time through the network.

The prediction function in step (3) is usually trained through machine learning. Consequently we need to assemble a training database for training of the model, consisting of training examples and target labels. From the point of view of the data set provider, the examples are the individual audio signals and their targets $\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}$. However, for training, the examples for the model are actually the segments $\dot{\boldsymbol{M}}^{(i,s)}$. Consequently, we

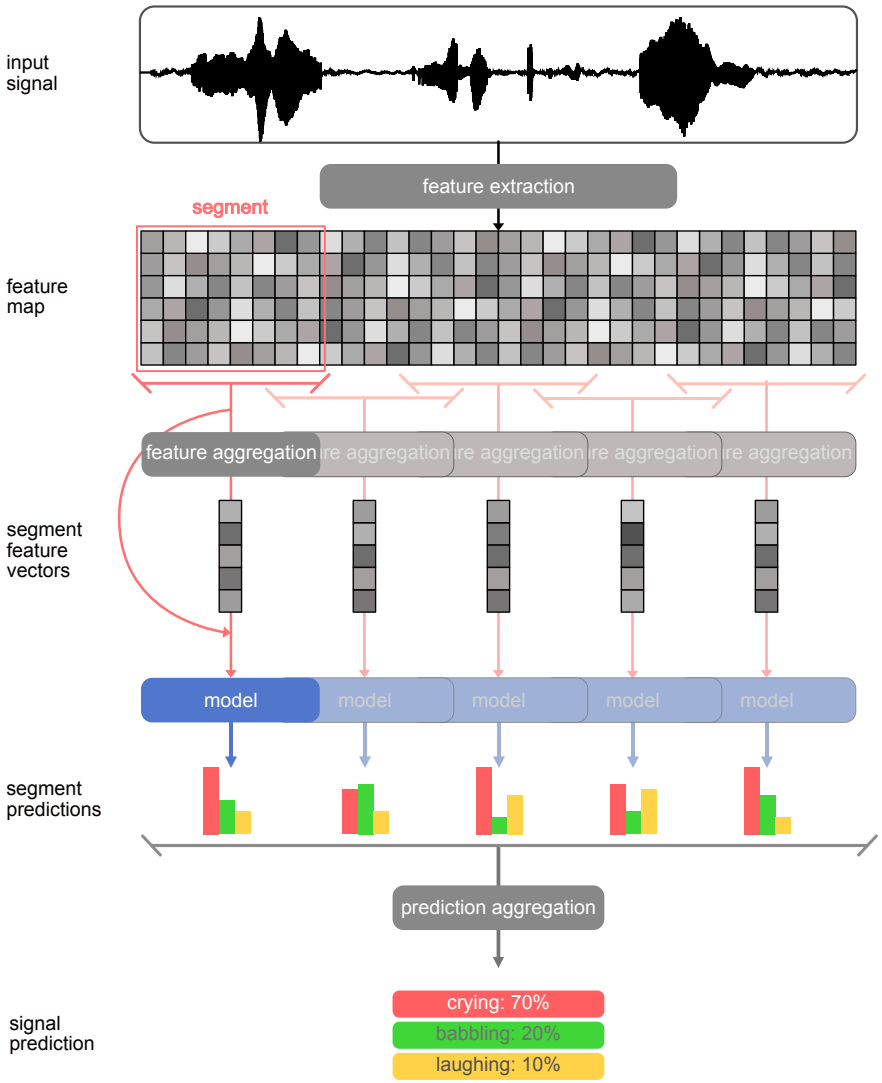

Figure 2.10: **Schematic visualization of the audio recognition process.**

need to determine a method to inherit the signal-level annotations onto the segments. In classification tasks, we might simply inherit the signal-level annotations onto all segments. However, this induces the implicit assumption that the target information is present in all segments, which might not be the case.

2.4.4 Evaluation of ASR system

Evaluation of an ASR system largely corresponds to evaluation of any ML model as described in section 2.2.4. However, one special consideration is the **album effect**: Machine learning evaluation assumes examples to be i.i.d., so that the test data actually is new compared to the training data. However, particularly acoustic data sets often

violate this assumption, causing a hidden information flow from training to test data that models might exploit unintentionally.

This hidden information flow occurs whenever the data set displays a correlation between acoustic features and the target variable that (a) is caused by recording conditions or context rather than recording content, and (b) is present in training and test data. For example, if in a classification task each class is recorded with a different microphone, the system might learn to differentiate the microphone characteristics rather than the actual target class characteristics.

Avoiding the album effect requires humans to make prior assumptions about which recording conditions might cause an album effect. These recording conditions must be noted as meta data. The exact variables that might cause an album effect are application specific. Mostly, signals are assumed to display context-related correlation if they are similar regarding the recording time, recording location or sound producing instance (e.g. speaker). Consequently, data set splits into training, validation, and testing must ensure data signal sets are mutually exclusive regarding such factors. [176, chapter 6]

CHAPTER 3

State of Research

The goal of any research endeavor is to advance the knowledge in its respective research discipline. The research disciplines that specifically are relevant to this thesis are the automatic recognition of acoustic communication events of (1) infants and (2) chimpanzees. However, these research disciplines are not isolated, but exist within the context of broader research disciplines and communities. Consequently, we have to consider the state of research of both levels: Primarily the state in the specific disciplines (automatic recognition of infant and chimpanzee acoustics) and secondary in the broader "parent disciplines".

The parent discipline relevant to this thesis is general ASR. Historically, research has primarily been driven through scientific publications in journals. However, in recent years conferences and competitions have considerably contributed to the advance in research, particularly for applied ML. A competition usually involves a set of challenges, such as "automatic classification of recordings of infant vocalizations into a set of predefined classes". A challenge typically provides a development and test set. Participants then design algorithms for solving the challenge and their submissions are evaluated on a test set regarding a predefined performance metric. The algorithm with the best test performance wins the competition.

The advantage of such competitions is that recognition algorithms are comparable under the same evaluation setup. In conventional research, such performance comparison might not be as straightforward, as various papers might use the same data set, but different evaluation setups. The disadvantage of this approach is that research might become overly focused on reaching high performance. Other research questions that are valuable as well but do not contribute directly to optimizing performance loose attention, e.g. investigating *why* certain approaches outperform others.

The goal of this chapter is to introduce the state of research relevant for studies (A) – (D). For study (A), this means the state of research in classification schemes for infant vocalizations. For studies (B) – (D), this means the state of research in automatic recognition of infant and chimpanzee acoustic communication, in the context of the

broader research developments in ASR. This context involves the developments in the relevant competition-driven research communities. Based on these states, I specify the research gaps.

The structure of this chapter is as follows. Section 3.1 introduces scientific competitions relevant to the studies of this thesis. Section 3.2 introduces the influence of deep learning on ASR, as this was of primary importance for my research questions. Section 3.3 presents the state of research specifically relevant to the studies and highlights the respective research gaps. This includes classification schemes for infant vocalizations, and automatic recognition of infant vocalizations and chimpanzee communication.

3.1 Related scientific competitions and communities

There are two annual competitions with primary relevance to automatic recognition of non-verbal acoustic communication events: The **Detection and Classification of Acoustic Scenes and Events** (**DCASE**) competition and the **Interspeech Computational Paralinguistics Challenge** (**ComParE**) competition.

The DCASE competition is associated with the equally named DCASE conference[1]. Both, the competition and conference, are focused on advancing the state of research on general ASR. This involves recognition tasks of all kinds of sounds, e.g. acoustic event detection of classes such as gun shots or sirens. The competition has been established as an annual, reoccurring event in 2016 (however, there was one first competition in 2013 with a subsequent hiatus). It is coordinated by the Audio Research Group of Tampere University in Finland.

Particularly in its first years, the competition involved challenges associated with the fundamental ASR tasks as defined in section 2.4.1. Challenges change each year, with one exception: The conference's "flagship" challenge is *automatic classification of sound scenes*, where scene classes are cafe, street etc. This is the only challenge to reoccur each year. In recent years, challenges have diversified and grown increasingly complex, deviating from the core focus of ASR tasks. For example, since 2019, they have been hosting a challenge on audio captioning and source separation. [106, 107, 109, 110, 111, 132, 176]

The ComParE challenge is associated with the annual INTERSPEECH conference[2]. It was established as an annual competition in 2009. The competition focus are challenges on **computational paralinguistics** (**CP**). The organizers define paralinguistics as "states and traits of speakers as manifested in their speech signals properties". Consequently, CP is the computational analysis of paralinguistics.

The challenge usually focuses on monophone classification tasks, as defined in section 2.4.1. Acoustic data sets and target classes are related to paralinguistics. A typical CP challenge is the automatic recognition of speech sentiment. As in the DCASE competition, all challenges usually change annually. With time, recognition challenges have

[1] `http://dcase.community`

[2] `http://www.compare.openaudio.eu`

diversified and deviated from the core focus of paralinguistics. For example, they hosted challenges on automatic recognition of heart sounds in 2018. [148, 150, 151]

Both of these communities are separate. However, they offer similar tasks, in a sense that they could generally be solved with similar algorithmic approaches. The ComParE community is more directly related to the particular recognition tasks of this thesis. As CP might be viewed as a special case of ASR, the DCASE community is related as well.

There are further research communities that might be considered relevant to automatic recognition of non-verbal acoustic communication as well. Examples include music information retrieval or speech recognition. However, I limited this discussion to CP and DCASE communities, as these affected the state of research relevant to this thesis' studies most directly.

Automatic recognition of animal sounds is the general area of research for study (D). However, as far as I know, there is no dedicated annual competition focused on this subject. Section 3.3.3 presents the related scientific literature.

3.2 Deep learning in ASR and CP

3.2.1 The deep learning boom

As mentioned throughout several sections in this thesis (sections 1.1.5, 2.2, 2.4), ASR systems have been based on ML approaches for a long time.

Conventional ML models, such as logistic regression, were long documented to reach poor performance when trained on raw signal inputs of visual or audio data, i.e. pixels of images or waveforms of audio signals (see foundations section 2.2.5 on logistic regression). The main reason was that such models can not incorporate the grid-structured topology of signals: Slightly shifting the signal leads the input signal to produce widely different response in the classifier, while to the human observer nearly no change occurred. [46, chapter 1,9]

Feature engineering used to be the main way to tackle this issue. The idea is to represent signals through a set of relevant features, instead of the entire raw signal. Each feature should capture a signal property hypothesized to be relevant for the task and exclude irrelevant signal properties. Such features are commonly referred to as *hand-crafted features*, as they were designed manually by experts. For example, when the challenge is speaker identification based on audio signals, we might design a feature to estimate vocal tract length. [46, chapter 1,9][176, chapter 5]

Solving a task with an **conventional ML approach** means we primarily focus on feature engineering and employ rather simplistic conventional ML models, such as logistic regression. The main challenge with this approach is that feature engineering is complex. Designing a feature describing the vocal tract length might be nearly as challenging as solving the actual task directly. Also, features tend to be task-specific, meaning that we would have to redesign at least some of the features for each new task. [46, chapter 1,9][176, chapter 5]

Representation learning is an alternative approach. Signals are left in their raw data representations, e.g. pixels in images. The model is designed to contain stages that are able to learn to extract feature representation themselves. **Deep learning** means that the model is made up of a deep hierarchical structure of layers, so that each layer learns concepts of increasing complexity. Typically, these are neural networks. [46, chapter 1,9][176, chapter 5].

The main challenge in representation learning systems is that they require far more data to be trained without overfitting. The reason is that models are essentially trained to perform two tasks at once: feature extraction and prediction onto the target domain. Consequently, they have more degrees of freedom and consequently a greater capacity, which increases the overfitting risk. [46, chapter 1]

Deep learning was primarily made popular in the scientific community through computer vision, particularly through the annual **ImageNet Large Scale Visual Recognition Challenge**[3]. This was an annual competition between 2010 and 2017. The competition comprised a reoccurring challenge on image classification into 1000 classes. In 2010 and 2011 the challenge was dominated by conventional ML approaches. In 2012 Krizhevsky et al. [85] won the challenge through a deep learning system based on a CNN by a considerable margin to the second placed system. The network architecture is commonly referred to as *AlexNet*.

Since then, the challenge has been dominated by deep learning systems based on CNNs. Some of the network architectures submitted by other participating teams in the subsequent years became popular: Deep learning frameworks, such as `tensorflow`, make these architectures, which were pre-trained on the ImageNet data set, accessible for other image recognition tasks. Examples for such networks are *VGGNet* from 2014 [155] and ResNet from 2016 [55]. Please refer to section 2.3.4 for an overview over AlexNet, VGGNet and ResNet.

3.2.2 From conventional ML to deep learning in ASR and CP

The success of CNN-based deep learning systems in computer vision has inspired a similar shift from conventional to representation learning in ASR. I explain this shift in the context of the general framework for audio recognition outlined in section 2.4.3.

- **Conventional sound recognition systems** usually employ *hand-crafted, complex* LLDs as features. The feature design process follows the process outlined in section 2.4.2. An example for such feature sets are GeMAPS [39]. Models are *simple* in a sense that they can not directly process two-dimensional feature maps, but only vector inputs. They either rely on temporal feature aggregation or directly predict each individual LLD frame. They usually are conventional machine learning models, such as logistic regression or tree models discussed in section 2.2.5.
- **Representation learning sound recognition systems** usually employ *simple audio representations*, i.e. the raw audio waveform or spectrogram types as dis-

[3] `https://www.image-net.org/challenges/LSVRC/`

cussed in section 2.1.1. There is an ongoing argument in the research community on whether spectrograms should be counted as simple representations or handcrafted features. The current consensus is to consider them as simple representations, as (1) they have served as a generic signal visualization tool for centuries independently of ASR, (2) they are unspecific to the task, and (3) there is evidence that the human auditory system performs a STFT-like operation as well for hearing. Models are *complex*, i.e. they usually are neural networks that are constructed to incorporate the grid-structured topology of the feature map, e.g. CNNs, RNNs or CRNNs. Consequently, they do not require temporal feature aggregation as defined in section 2.2.5. [177][176, chapter 3]

The development from conventional sound recognition systems to representation learning systems can be tracked in the baseline systems of the ComParE and the DCASE competitions:

The ComParE competition used the same baseline systems for all challenges in each respective year (at least to this date). Starting in 2014, the challenge featured one baseline system, which was a conventional sound recognition system. The *OpenSmile* system extracts 64 LLDs, comprising features such as root mean squared energy, zero-crossing rate, spectral energies, fundamental frequency etc. All of these features require elaborate analysis of the signal to be computed. Then, LLDs are aggregated through so-called **functionals**, i.e. statistical summaries across time, as described in the *feature aggregation step* in section 2.4.3. The resulting feature set comprises 6373 features per analyzed time segment. The discriminating model is a support vector machine (SVM), which is a linear model similar to logistic regression. The system has been a baseline system for the ComParE challenge since 2012 up until today [149, 184].[4]

In 2017, an end-to-end system was introduced as an additional baseline system, referred to as End2You. This system retains signals directly as time domain waveforms. The waveform is fed into a neural network, which is a one-dimensional CRNN. The model uses two blocks of convolutional and max-pooling layers, with 2 subsequent LSTM layers. [172, 174].

The DCASE competition generally uses different baseline systems for each challenge. As most challenges change annually, I use the *automatic classification of sound scenes* challenge as a reference point, as this is the only reoccurring challenge. In 2016, the baseline system was conventional: The LLDs were MFCCs, delta coefficients, and acceleration coefficients. The classifier was a Gaussian mixture model (GMM). MFCCs are features commonly used in automatic speech recognition. They are produced by applying a second Fourier transformation onto the power spectrum. GMMs are models that model the likelihood of features for each class as a mixture of Gaussian distributions and infer the class through the Bayesian rule [106].

In 2017, the feature set was a Mel-scaled spectrogram and the model was a FCNN with two layers. It is a hybrid between conventional and representation-learning systems, as the feature set is simple, but the model requires feature aggregation by flattening

[4]The system is also referred to as the ComParE feature set + SVM approach.

the input feature map [108]. The 2018 and 2019 editions introduced a deep learning baseline system. The system used Mel-scaled spectrograms as feature maps and a CNN as classifier. [110, 132]

The DCASE and CP communities differ on how prevalent deep learning based systems have become. In the DCASE, deep learning systems have completely replaced conventional ML approaches: In 2017, approximately two thirds of the participant's entries for the scene classification task employed deep learning [109] and all of the top placed systems were deep learning ones. In 2018, nearly all participant systems were deep learning systems[5].

In CP, deep learning based systems also have become popular, however to a lesser degree. In 2017, half of the participating systems were deep learning systems [177]. From 2018 until today, the challenge still hosted at least two baseline systems of the conventional type, in addition to deep learning baseline systems [150, 151, 152]. As opposed to the DCASE challenge, conventional systems have still been performing on par with deep learning systems and are still submitted by participants (on average, details differ regarding the specific challenge).

Consequently, the shift to deep learning systems in ASR occurred with a delay of about 5 years to the computer vision community, as reflected by the developments in the DCASE and CP communities. One of the major reasons is data set size: Collecting and labeling large amounts of sound data is more time-consuming than it is for image data. Benchmark data sets for ASR are usually a fraction of the size of benchmark image data sets, such as the ImageNet data set. [176, chapter 6].

At the moment, deep learning has become the default choice in the DCASE community. However, In CP, deep learning and conventional approaches are coexisting. One of the major reasons for this again is data set size: CP usually offers small data sets, as it requires specialized recording procedures and experts for labeling. DCASE challenges can use data sets from free internet sources, such as `freesound.org` or YouTube. [176, chapter 6][148, chapter 6]

3.2.3 Spectrograms in representation learning systems

Combining spectrograms with CNNs has increasingly become popular for classification tasks in ASR. Essentially, the spectrogram is merely viewed as a gray-scale image, so that the CNN performs "image classification". The CNN architecture is either directly adapted from one of the popular CNNs for image recognition (e.g. those presented in section 2.3.4) or one that at least follows similar design principles. This type of approach has become so prevalent that I refer to it by its own name, the "**spectrogram-as-an-image+CNN**"-approach.

The core principle in this is that the CNN treats the time and frequency axis of the input spectrogram equally, e.g. through quadratic convolutional layers, making it "unaware" of the fact that the dimensions of the spectrogram carry different meanings;

[5]For the 2018 DCASE edition, there is no overview paper. See `http://dcase.community/challenge2018/index` for an overview of all participant systems.

it merely considers the input as a grid of spatially arranged values. To the best of my knowledge, Piczak [128] was the first one to apply this approach to an ASR task in 2015, which was scene classification, and inspired the usage in further DCASE challenges. Research teams then began to directly adapt architectures on computer vision for ASR, e.g. AlexNet, VGGNet, ResNet etc. (see foundations section 2.3.4 on these architectures). Hershey et al. [60] from Google tested such architectures for large-scale sound classification for youtube videos in 2017 and reached satisfactory performance. The great majority of entries to classification challenges in the DCASE competition since 2017 uses this approach as well [80, 107, 109, 110, 165]. Sections 3.3.2 and 3.3.3 discuss further examples of this approach type applied infant vocalizations recognition and chimpanzee recognition specifically.

The most common spectrogram variant is the log Mel-scaled spectrogram (see foundations section 2.1.1.3 on this spectrogram type) [26, 36, 42, 56, 60, 84, 108, 110, 132, 153, 169, 177]. Cramer et al. [32, p. 2] stated that "when a sound is pitch-shifted the pattern created by its harmonic partials change when using a linear frequency scale, whereas with a (quasi) logarithmic frequency scale such as the Mel scale, pitch shifts result in a vertical translation of the same harmonic pattern, meaning that convolutional filters should generalize better when using the latter".

In section 2.4.3 I introduced an approach to detection, by subdividing the input feature map into shorter segments and classifying each segment individually. However, if we aim to achieve fine-grained resolution for detection outputs, we need to either reduce segment size or increase the overlap between segments. The former might reduce the recognizability of classes and the latter is computationally expensive. Therefore, CRNNs have increasingly gained popularity for detection challenges specifically. Recurrent layers are able to output predictions for each time frame of the input feature map. This way, they can incorporate larger temporal contexts and output predictions for every LLD. The technique was primarily popularized by Cakır et al. [22] and has subsequently been adapted for other audio detection challenges.

3.3 States of research specific to studies (A) – (D)

3.3.1 Classification schemes for infant vocalizations – study (A)

Section 1.1.3 introduced the importance of infant vocalization classification schemes for various infant assessment scenarios.

There currently is no consensus on one single of such classification schemes for infant vocalizations. There is a multitude of classification schemes that were proposed for different purposes.

Buder et al. [21] presented a catalogue-like classification scheme, in the context of assessment of infant vocal development. The classification scheme operates on the breath group unit segmentation criterion (see foundations section 2.1.2). It differentiates between so called *reflexive sounds* and *protophones*. Reflexive sounds involve cries and fusses, laughs, and vegetative sounds (breathing, burping, coughing, sucking etc.). Pro-

tophones are all other vocalizations. These are hypothesized to be precursors to speech. Examples for protophones are vowels, quasivowels, squeals, raspberries, etc. Protophones also involve sequences of repeated sounds, such as marginal or canonical babbling (these correspond to what the layperson usually imagines as stereotypical babbling).

Further classification schemes in the literature are part of assessment tools, in which infant vocalization classification is an intermediate analysis step.

The most elaborate assessment tools are for assessment of vocal development in infants. Here, the emergence of vocalization classes is associated with expected ages. The common denominators of these classification schemes are that (1) they put an emphasize on protophone vocalizations, and (2) they either use unit segmentation criteria (A) or the breath group criterion (see foundations section 2.1.2). Nathani et al. [117] proposed the SAEVD-R, which is among the currently most prevalent of these tools. The set of vocalization classes is similar to the ones presented by Buder et al. [21], however associated with typical ages of emergence. Oller [121] presented a similar assessment tool, which might be considered to be a predecessor to the tool of Nathani et al. [117]. Stark [163] also presented an assessment tool, which put more focus cry and fuss vocalizations, as they hypothesized crying to also play an important role in vocal development.

Pain scales are assessment tools for infant pain assessment. As introduced in section 1.1.3, they associate the occurrence of vocalizations with pain scores. Neutral and positive vocalizations of any kind usually form one category signaling no pain. Negative vocalizations such as moaning, whining, fussing, wailing, crying, screaming etc. form an ordinal scale to signal pain intensity. These classification schemes usually omit the segmentation criterion or further definitions of vocalization classes, i.e. they rely on internal guidelines by the paediatric staff for reliable usage. [5, chapter 6][104, chapter 5]

There are further classification schemes presented by research papers for a variety of research goals. Scheiner et al. [145] investigated developmental and emotion-related changes in infant vocalizations. For this, they defined a classification scheme of 12 call types. Lin and Green [93] studied the changes in infant expressive behavior to mothers, by assessing occurrences of negative, neutral and positive vocalizations.

All presented classification schemes are aimed at trained professionals in infant vocalization assessment. There is little research on the assessment capability of laypersons on infant vocalizations. Studies that employed laypersons for vocalizations assessment did so primarily for rating of distress vocalizations (i.e. fuss- and cry vocalizations). Xie et al. [187] asked parents to rate the level-of-distress of cry-like signals on a continuous scale. The goal was to identify correlations between the level-of-distress and acoustic properties. Barr et al. [15] asked parents to document their infant's crying behavior in dictionaries. They were asked to discriminate *content*, *fussing* and *crying*. These studies defined crying as strings of continuous vocalizations, i.e. as sequences in the framework presented in section 2.1.2.

There are many open research questions on assessment of infant vocalizations through classification schemes:

- There is no consensus on classification schemes. There are certain vocalization classes that carry the same name across classification schemes, such as *crying*.

However, these vocalizations are defined differently in detail. For example, some studies define crying as a unit, while others define it as a sequence; some differentiate fussing and crying, while others refer to any distress vocalization as crying. [16, chapter 11]

- There is little research on the capability of laypersons to classify infant vocalizations. All of the above mentioned scenarios on vocalizations assessment and classification were aimed at professionals. This is of importance for any application in which this capability is of interest, e.g. employing laypersons for crowd-sourcing data set labeling.

3.3.2 Automatic recognition of infant vocalizations – study (B) and (C)

Section 1.1.4 introduced the advantages of supporting infant vocalizations assessment through automatic systems. Section 1.1.3 introduced that, ideally, we want a system that (1) detects infant vocalizations, and (2) classifies them into a discrete set of vocalizations. This allows to employ the system for monitoring and connect the system output to a variety of subsequent analysis applications.

The LENA[6] system most closely fulfills this requirement. Oller et al. [123] originally presented it in a study. The system automatically detects infant vocalizations and subsequently classifies them into one of three classes: *cry*, *vegetative vocalizations* (laughter, coughing, etc.) and *speech related vocalizations* (protophones). The analysis goal is monitoring of vocal development, with further specific purposes, such as early detection of speech impairments. The LENA project is a non-profit organization that distributes devices and software.

The ComParE competition 2018 hosted a challenge on the automatic classification of infant vocalizations [150], named the *crying* subchallenge. The challenge was to classify general infant vocalizations into the classes *neutral*, *fussing*, and *crying*. Seven teams participated and there were four baseline systems. This challenge was of particular importance to the research in automatic infant vocalization recognition, as it was the first time to allow comparing various approaches under the same rules. However, the challenge did not completely fulfill the above mentioned requirement for monitoring systems, as it did not involve prior detection of vocalizations.

The majority of the remaining research is aimed at infant crying specifically. The research is fragmented regarding recognition goals and methods. Examples of recognition goals are: (1) automatically deriving the crying cause by analyzing the cry signal, e.g. deriving whether the reason is sleepiness, hunger, or pain [1, 23, 24, 25, 112, 119, 126, 127, 137, 175][7], (2) automatically deriving pathologies, such as asphyxia [37, 43, 125, 131, 139, 178, 190, 191], and (3) automatic regression of the level-of-distress or pain

[6] `https://www.lena.org/technology/`

[7] I highlight that the validity of these research endeavors might be questionable, as it has repeatedly been shown that humans are unable to distinguish the infant crying reason without contextual information [16].

intensity based on analysis of the cry signal [157, 187]. All of the proposed algorithms lack practical applicability, as they require limiting inputs to crying specifically.

Lastly, there is research on automatic detection of cry vocalizations [30, 87, 115, 154, 189]. Such systems are applicable for long-term monitoring, however are aimed at detecting crying specifically and are therefore more constrained regarding specific applications.

Regardless of the specific recognition challenge, automatic recognition of infant vocalizations experienced increased interest in deep learning methods, analogous to the global trend in ASR.

The great majority of systems for automatic infant vocalization recognition employed conventional ML approaches, as defined above. This applies to the previously mentioned LENA system, which uses a similar approach to the DCASE 2016 baseline system [106] with GMMs. Most of the remaining research employs MFCCs or LPCCs as features, and used conventional ML models, such as SVMs [23, 25, 137, 175], k-nearest neighbor [30, 125], decision trees [37], hidden-markov models [115, 187], FCNNs [127, 178, 190, 191]. The studies [43, 119, 139] did in-depth comparative studies on various of such conventional ML algorithms.

There is less research on deep learning for infant vocalization recognition due to its novelty. The *spectrogram-as-an-imge+CNN* approach gained popularity as well. Chang and Li [23] and Lavner et al. [87] did preliminary studies for identification of the crying cause and cry detection, respectively, by using a single custom CNN architectures. Yao et al. [189] applied AlexNet [85] for cry event detection and reported significant performance gains over the LENA algorithm. Severini et al. [154] tested single-channel and multi-channel recordings for cry detection with custom CNNs.

The ComParE Crying sub-challenge 2018 also featured the full spectrum of types of systems [150]:

Conventional systems in this competition were as follows: The first was the baseline system *Open SMILE*, presented in section 3.2). The second baseline system *OpenXBOW* [184] used a Bag-of-words approach to feature generation and inputs them to an SVM classifier. The participants Gosztolya et al. [48] proposed various modifications of the *OpenXBOW* baseline, one of which applies FCNN outputs for the Bag-of-words codebook generation.

Deep learning systems were as follows: The baseline system *End2You* [174] feeds raw waveforms into a CRNN. The baseline system *AuDeep* and the one of Syed et al. [169] combined Mel-scaled spectrograms with CRNNs. Turan and Erzin [173] proposed a CNN derivative named *CryCaps* with spectrogram inputs, but only indicated validation set performance. Wagner et al. [177] presented an in-depth comparison of various feature sets combined with an RNN. Some of these sets were simple spectrograms and some were more elaborate, such as the COMPARE feature set [184]. They found that hand-crafted LLDs outperformed more simple spectral representations. The remaining entries likewise combined hand-crafted LLDs with RNNs [67, 197] .

The highest single system performance was held by the conventional *Open SMILE* [38] system with 73.2 % UAR. Consequently, none of the deep learning systems managed to

outperform this conventional one. This observation is in accordance to the general developments in CP discussed in section 3.2.2.

Based on this state of research, I identified two major research gaps in automatic recognition of infant vocalizations:

- While the application of conventional recognition systems has received ample research, there still is a lack of in-depth studies for representation learning and deep learning systems. Such research has been restricted to singular performance indications of fixed pipelines.
- There is a lack of systems designed for practical applicability to monitoring scenarios with various assessment goals in mind. Previously proposed systems have either been focused on crying in particular or focused on classification instead of detection.

3.3.3 Automatic recognition of chimpanzee communication – study (D)

Automatic detection of chimpanzee calls is of primary importance for automatic monitoring of wild chimpanzee populations. There is a multitude of monitoring applications, from assessing chimpanzee home ranges for behavioral studies to early-warnings systems for areas with human-wildlife conflict. [74, 75]

Passive acoustic monitoring (**PAM**) is one of the most widely employed methods for monitoring wild animals. Here, **autonomous recording units** (**ARUs**) are distributed over an area for constant soundscape recording. The main advantages of PAM are: (1) minimal intrusion, as humans are only required for installation and maintenance of devices, (2) sampling over large spatial and temporal scales, and (3) detection of animals in habitats where visual recognition is limited, e.g. dense rain forests. However, PAM also produces large amounts of audio data that quickly become infeasible to curate manually by humans. Consequently, algorithms for automatic detection of target species are of primary importance for PAM settings. [75]

Ideally, the goal is to design a detection system to detect various chimpanzee vocalizations of interest. To my knowledge, there are only two publications that specifically focused on this task: (1) Heinicke et al. [57] investigated an automatic system for detecting calls of various primate species in PAM recordings of a tropical forest. Their algorithm employed a conventional ML approach based on hand-crafted features and GMMs. Algorithm performances varied with respect to target species and call type, from 10 % $F1$ for Diana monkeys and King colobus monkeys to 4 % for chimpanzee drumming and 0.2 % for chimpanzee vocalizations. (2) The master thesis of Dev [35] also investigated detection of chimpanzee calls, however examples of the negative class were made up of the Urbansound8K [144] data set classes (e.g. car horn or gun shot), instead of naturalistic background sounds.

In recent years, deep learning has gained increasing popularity for animal acoustic detection as well. Here, the *spectrogram-as-an-image+CNN* approach gained popularity

(see section 3.2.3 on this approach), i.e. all of the following related literature fits into this approach. Bergler et al. [17] applied variants of the computer vision CNN architecture ResNet [55] for detection of orca calls in long-term recordings (see foundations section 2.3.4 on ResNet); Bjorck et al. [18] applied the computer vision CNN architecture DenseNet Huang et al. [65] for detecting African forest elephants with PAM; Oikarinen et al. [120] applied siamese CNNs with stereo inputs to the detection of various marmoset monkey calls. Mac Aodha et al. [98] investigated CNNs for bat detection. The DCASE competition 2018 hosted a challenge on automatic bird detection as well. The baseline system [165] employed custom CNN architectures. The winning team Lasseck [86] of the challenge applied ResNet [55] with extensive data augmentation, similar to Bergler et al. [17].

However, there are still research gaps in automatic primate call detection as well as automatic animal call detection in general.

- **Consideration of target class rarity in PAM settings:** The majority of recordings in long-term PAM recordings will comprise background noise rather than target calls [18, 57, 120]. Consequently, databases feature a heavy imbalance between background and target class. Numerous studies showed that class imbalances can have detrimental effects on automatic system performances, as classifiers are usually biased towards the majority class [53, 72, 183]. However, all previously mentioned studies worked with databases with strongly reduced amounts of background samples, biasing the class distribution towards the positive class. This bias was either already present in the respective database or produced by the authors through discarding fixed percentages of noise samples. Additionally, nearly all papers measured system performances with metrics unsuited for unbalanced settings, such as accuracy (see foundations section 2.2.4.1 on accuracy). These give overly optimistic results in recordings with heavy class imbalance as they are biased towards the majority class [72].
- **Time-continuous detection:** All previously mentioned deep learning systems approached detection challenges by classifying broader spectrogram segments of various seconds, e.g. 25 second patches [18]. Section 2.4.3 discussed the advantage of using CRNNs with time-continuous outputs instead. However, these networks have not yet been applied to animal detection tasks.

CHAPTER 4

Study (A): Investigation of the Assessment of Infant Vocalizations by Laypersons

4.1 Study goal

The central goal of this study was the development of a classification scheme for infant vocalizations based on the discrimination capability and assessment behavior of laypersons. This study focused on human sound perception, *not* on automatic recognition of infant vocalizations. The precise research questions were:

- What are the salient labels used for naming infant vocalizations?
- What is the relationship between those labels?
- How do labels relate to affect assessment, such as valence?
- Which classification schemes can be derived based on these salient labels and their relationships?

In the context of training ML algorithms for automatic infant vocalization recognition, such classification schemes are of particular importance for labeling of datasets through crowd sourcing. In the context of vocalization assessment, it is of importance to better understand the general assessment capability of untrained listeners, which has not yet been investigated in detail. This relates to the research gaps discussed in section 3.3.1.

The methodology for this study followed the *Nijmegen Protocol* [159], a method for uncovering linguistically expressed classes and taxonomies. This involved a survey that presented participants with a set of acoustic stimuli, which they rated regarding textual labels and continuous affect scales. I derived the salient labels and their affect ranges

by aggregating ratings across participants for each stimulus. Additionally, I investigated the relationship between labels by analyzing intra-rater similarities of stimuli based on label and affect ratings. Finally, I derived taxonomies as semantic maps based on the discovered salient labels and their relationships.

This study interconnects with study (C), as the classification scheme derived through this study was applied for labeling the data set for the automatic recognition algorithm.

4.2 Materials and methods

The methods section is structured as follows: Section 4.2.1 introduces the reference approach this study was based on. Section 4.2.2 presents the acoustic data set. Section 4.2.3 presents the survey procedure. Section 4.2.4 presents the survey rating items in detail. Section 4.2.5 summarizes the participants of the study. Finally, section 4.2.6 presents the analysis methods applied to the participant ratings.

4.2.1 Study framework: Nijmegen Protocol

The study followed a method that is referred to as the **Nijmegen Protocol**. This is a survey-based approach for the discovery of linguistically expressed classes inside a domain. Majid et al. [99] first applied this method to identify universally recognized classes of material destruction, such as *cutting* and *breaking*. Slobin et al. [159] refined the method and proposed the term *Nijmegen Protocol*. They applied it to identify classes for manners of human gait, such as *running* or *crawling*. Anikin et al. [12] applied it to auditory assessment of human non-linguistic vocalizations and determined four basic call types *laugh*, *cry*, *scream* and *moan*. That study primarily inspired the application of the framework here, as it shares similar goals, however applied to infant vocalizations instead of vocalizations from adults. I summarize the method as follows:

1. **Stimuli selection:** First, a set of stimuli is selected. They should be representative for the variability inside the investigated domain.
2. **Rating:** Participants rate stimuli in a survey. In this method, the primary rating items are textual descriptions, i.e. labels. Some studies let participants provide free textual descriptions [159], while other studies used predefined, finite label sets [12]. Some studies included additional rating items, such as affect assessments [12].
3. **Calculation of stimuli similarities:** For each pair-wise combination of stimuli a similarity score is determined. A similarity score is calculated for each participant individually and then averaged across participants. This means that similarities measure the average intra-rater similarity, *not* the inter-rater similarity. This allows participants to speak different languages and consequently discover cross-cultural classifications. The resulting distance matrix is then applied to unsupervised learning algorithms, such as clustering or dimensionality reduction algorithms. [12, 159]

4.2.2 Acoustic data set

I selected stimuli representing *sequences*, rather than *units* (see foundations section 2.1.2 on these terms). Most previous studies on infant vocalizations operated on *utterances*, i.e. units with unit segmentation criterion (A). This is particularly common for studies on infant vocal development [12, 21, 51, 117, 124, 145]. The most prevalent segmentation criterion in these studies is the breath group (see foundations section 2.1.2).

However, particularly studies on perception of cry-like infant vocalizations showed that human listeners incorporate acoustic information spanning various units, such as the rhythm of units [16, 52, 185, 187, 193]. I additionally hypothesized that laypersons would be me comfortable by judging short collections of units, rather than units in isolation.

Therefore, I aimed stimuli to be recording excerpts of approximately 6 s spanning various units, similar to *sequences* as defined in section 2.1.2. However, if a unit happened to be present in isolation in a recording, i.e. surrounded by a long pause, it was also considered, including some of that pause.

I collected audio recordings of infant vocalizations from freely available online sound libraries. The sources were:

- `https://freesound.org`
- `http://www.bigsoundbank.com`
- `http://www.soundarchive.online`
- `https://www.zapsplat.com`
- `https://www.soundbible.com`
- `https://www.soundjay.com`

All recordings using the keywords *baby* or *infant* were considered. I discarded recordings that met at least one of the following criteria:

1. Insufficient recording quality: This included constant and high amounts of background noise or sounds, excessive reverberation and echo, sound effects etc.
2. Shorter than 4 s: These recordings provided insufficient temporal context according to the discussion above.
3. Infants older than 9 months of age: Infant vocalizations increase in complexity with age. According to the SAEVD-R [117] infants enter the vocalization stage *advanced forms* at 9 months of age. I excluded this stage to limit the complexity of the *babbling*-class. The age was determined through the description of the online source. If the description was absent, I determined acoustically whether a signal contained such advanced form of babbling as described in [117] and excluded such signals.

The resulting database consisted of 228 sound files with a total duration of 7910 s and a mean duration of 34 s. I extracted segments from these recordings as stimuli

candidates, aimed at a length of 4 – 8 s according to the discussion above. Segments had to contain at least one vocalization, even if it was just slight breathing, i.e. I did not extract segments with complete absence of vocal activity. I allocated segments to contain units that were acoustically consistent. Units were contained completely in segments, i.e. not cut across.

The number of resulting segments was 883, with a mean duration and standard deviation of 6 s ± 0.8 s, and a total duration of 5217 s. The set of segments displayed a wide variety of vocalizations, ranging from stereotypical vocalizations such as babbling or crying to barely audible vocal activity, such as quite breathing.

The pool of segments had to be reduced to a reasonable amount for participant rating. To ensure that the drawn sample displayed sufficient acoustic diversity, I evaluated each segment regarding the affect dimensions *valence* and *arousal* as defined by Russell [142] with continuous scales. I clustered segments based on these ratings through the k-means algorithm [14] with 10 clusters. I randomly sampled 10 segments from each cluster.

The resulting set contained 100 segments, which were the stimuli used for this study.

4.2.3 Procedure

The survey was provided through a customly programmed web site. Participants conducted the survey at home.

Figure 4.1 shows the survey main window for stimulus rating. Participants were presented with one signal at a time. The top part presented a stimuli's wave form with a progression marker, similar to digital audio workstations. Participants could play and repeat the signal autonomously through buttons. The middle panel contained the rating items: Three continuous rating scales **Stimmung** (**valence**), **Wachheit** (**energetic arousal**) and **Ruhe** (**tense arousal**); and one list **Bezeichnung** (**label**) for providing a textual description. Section 4.2.4 describes the rating items in detail. The bottom panel contained a button for proceeding to the next signal. There was no time limit for item rating. Additionally, participants could save, quit and resume rating sessions, i.e. they could distribute completing the survey over various sessions.

The first page of the survey presented introductory information. This comprised the survey goal, survey procedure and usage of the rating items. Section 9.1.1 states the instructions on the rating items. The second page listed five signals with rating items for participants to get used to the rating procedure. Participants were informed that answers to these signals were discarded. The warm-up signals were preselected to cover diverse and stereo-typical infant vocalizations, e.g. stereo-typical crying and babbling.

At the end of the survey, participants were asked about their personal data: gender (female or male); age (5-year intervals of 15 – 19, 20 – 24, ...); whether they were parents; whether they had professional working experience with infants such as baby sitting. For each of the four rating scales, participants should indicate how challenging they experienced their usage on a scale from 1 – 5, labeled with *very easy*, *easy*, *modest*, *hard* and *very hard*.

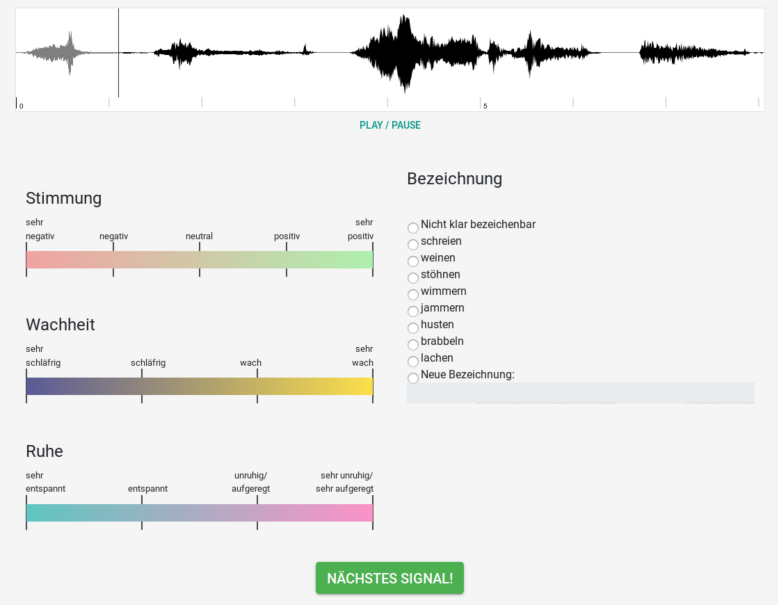

Figure 4.1: **Survey main window for stimulus rating.** Details are provided in the text.

4.2.4 Rating items

4.2.4.1 Label

The rating item *label* asked for a textual description of the vocalization, i.e. naming the acoustic category. I provided a starting pool of labels that could be expanded freely through custom labels.

The starting pool was as follows (English translations are provided in parentheses): schreien (screaming); weinen (crying); jammern (whining or fussing); wimmern (whimpering); stöhnen (moaning), brabbeln (babbling); lachen (laughing); husten (coughing); nicht klar bezeichenbar (no clear label)[1]. The former five terms originate from the only validated German pain scales KUSS [195, chapter A3] and BPSN [28]. The terms *babbling*, *laughing*, and *coughing* were determined in a pilot study, were all participants entered these terms. The option *no clear label* was provided for cases where participants felt that none of the starting labels applied, but neither could think of a fitting custom label. The introductory text encouraged raters to rather think of new labels than use this option.

Response options were provided via a radio button list. There was no preset value to encourage deliberate choices. Only one term could be chosen. Participants were instructed to select the more salient label if they thought to hear the presence of more than one.

[1]The stated English translations are meant to be literal translations, not semantically equivalent translations. For example, it might be argued that *crying* and *schreien* are closer semantically.

New labels were added through a text field. They were permanently added to the respective participant's label list, so they could be accessed later in the study analogous to the starting pool labels. Added labels were restricted to verbs consisting of a single word, i.e. words with the suffix *ing* (German: *en*) without spaces nor special characters. I imposed this restriction to promote monolexemic, elementary call types, rather than complex descriptions of the situation, similar to the study of Anikin et al. [12].

The introductory text highlighted that starting pool labels did not have to be used exhaustively. Participants could ignore starting pool labels that they felt were redundant or not present throughout the survey.

The final summary page asked participants to indicate whether the starting pools contained **synonym pairs**, i.e. labels that were redundant or particularly hard to differentiate throughout the study. I indicated that these labels would be mapped together for the respective participants.

4.2.4.2 Affect dimensions

The remaining rating items were three continuous scales for rating of the estimated infants' affect. I adapted the affect model of Schimmack and Grob [146], Schimmack and Rainer [147], which comprises the following scales: Valence (also known as pleasure) ranging from pleasant to unpleasant, energetic arousal ranging from sleepy to awake, and tense arousal ranging from calm to nervous. I chose to include affect ratings additional to the labels for the following reasons:

- Other studies that employed laypersons for rating of infant vocalizations most commonly required continuous ratings, rather than categorical descriptions. These studies were limited to rating of cry-like vocalizations and used the dimension *level-of-distress* [185, 187]. However, my study encompassed the entire domain of infant vocalizations rather than just cry-like vocalizations. Therefore, I employed a general affect model. I first tested the model by Russell [142] in a pilot study, containing just two dimensions: valence and arousal. However, participants raised concerns about the arousal dimension being ambiguous. Therefore, I split the arousal dimension into tense and energetic arousal as proposed by Schimmack and Grob [146] [147].
- Any correlations between affect and label ratings are exploitable for designing classification schemes, as they indicate another level of stimuli similarity that might not be obvious from label ratings alone. Particularly cry-like vocalizations have repeatedly been shown to be a *graded signal* [16, chapter 2]. In the most extreme case, affect ratings and labels could turn out to be completely redundant. Anikin et al. [12] discovered a strong but not perfect association between call types and affect ratings in a similar study.

I implemented affect ratings as continuous scales (see Fig. 4.1) with 101 ticks.

- **Valence**, labeled *Stimmung* had a range of $[-50, 50]$, with highlight labels *sehr negativ* (very negative), *negative*, *neutral*, *positiv*, *sehr positiv* (very positive) at ticks $-50, -25, 0, 25, 50$, respectively.
- **Energetic arousal** labeled *Wachheit* had a range of $[0, 100]$ with highlight labels *sehr schläfrig* (very sleepy), *schläfrig* (sleepy), *wach* (wake), *sehr wach* (very wake) at ticks $0, 33, 66, 100$, respectively.
- **Tense arousal**, labeled *Ruhe* had a range of $[0, 100]$ with highlight labels *sehr entspannt* (very relaxed), *entspannt* (relaxed), *unruhig/aufgeregt* (restless/excited), *sehr unruhig/sehr aufgeregt* (very restless/excited) at ticks $0, 33, 66, 100$, respectively.

The introductory text highlighted that scale ticks were only for orientation and that values could be chosen continuously. There were no preset values to encourage deliberate value choices.

4.2.5 Participants

A total of 23 persons participated. The gender distribution was 12 male and 11 female. The age range was 20 – 59, with a median age of 25 – 29. 7 of the participants were parents and 4 had professional working experience with infants. All participants completed the survey from start to finish.

4.2.6 Analysis methods

4.2.6.1 Post-processing of label ratings

Postprocessing meant manipulation of participant ratings after survey data collection was completed in preparation for the analysis. I applied the following steps for the label ratings:

- I fixed obvious spelling mistakes of newly added labels. In some cases, this resulted in labels that originally were different across raters to be mapped to the same label.
- If participants indicated *synonym pairs* at the end of the study, I unified those labels for each participant to the one she or he used more frequently in the survey.

4.2.6.2 Measurement of stimuli distances

I represented participant ratings of stimuli through matrices $\boldsymbol{R}^{(L)}, \boldsymbol{R}^{(V)}, \boldsymbol{R}^{(E)}, \boldsymbol{R}^{(T)}$, where the upperscript (I) indicate rating items (e.g. L = label, V = valence, E = energetic arousal, T = tense arousal). $R_{p,i}$ is the rating of participant $p \in \{1, \ldots, P\}$ for item $i \in 1, \ldots, 100$, where P is the amount of participants. Based on these I calculated distance matrices for stimuli $\boldsymbol{D}^{(L)}, \boldsymbol{D}^{(V)}, \boldsymbol{D}^{(E)}, \boldsymbol{D}^{(T)}$, where $D^{(I)}_{i,j} \in [0, 1]$ indicates the distance between stimuli i and j for the respective rating item. Consequently, $\boldsymbol{D} \in [0, 1]^{100 \times 100}$.

The label based distance $\boldsymbol{D}^{(L)}$ was calculated as

$$D_{i,j}^{(L)} = \frac{1}{P} \sum_{p=1}^{P} I(R_{p,i}^{(L)}, R_{p,j}^{(L)}), \tag{4.1}$$

where I is the indicator function. Consequently, this distance measures the percentage of participants that *individually* provided the same label.

For the affect rating items I calculated distances as

$$D_{i,j}^{(C)} = \frac{1}{P} \sum_{p=1}^{P} \frac{|R_{p,i}^{(C)} - R_{p,j}^{(C)}|}{S_{p,C}}, \tag{4.2}$$

where $C \in \{\text{valence, energetic arousal, tense arousal}\}$ is the scale index and $S_{p,C} = \max_p(R_{p,i}^{(L)}) - \min_p(R_{p,i}^{(L)})$ is the range for scale C of participant p.

When combining various items into distance calculation, I summed item-based matrices.

In the context of the Nijmegen Protocol, this calculation of the label-based distance corresponds to the established approach of counting equal descriptions [12, 159]. In the context of unsupervised machine learning, the approach corresponds to calculating distances through the Gower coefficient [49]. This distance metric is specialized in mixed-type variables, where nominal variables (label) employ the ordinary discrete metric and continuous variables (affect scales) employ the range-normalized L1-distance. Each rater and item is treated as a unique feature.

4.2.6.3 Aggregation of stimuli ratings across raters

One step in analysis of results was determining a single **consensus rating** for each stimulus and rating item. To accomplish this, I aggregated ratings across participants for each stimulus and item as follows:

For affect scales, the consensus was the average across all participant ratings.

For the label item, I chose a **most salient label** as the one most often voted for the respective stimulus. All labels from the starting pool counted as their own *voting label*. However, as participants often added similar, but not completely identical labels, I accumulated the number of newly added labels for each stimulus to represent a single *voting label*. If this accumulated count was chosen more often than starting pool labels, I chose the one most participants entered identically (including correction of spelling mistakes).

4.2.6.4 Measurement of association between label and affect ratings

I employed two methods for measuring of the association between label and affect ratings:

- **Predictability of most salient label:** I tested whether aggregated affect ratings allow precise prediction of the most salient label. I modeled the conditional probability $p(\text{label} \mid \text{emotion})$ through a multinominal logistic regression model and

assessed the goodness of fit through the UAR metric to account for class imbalance (see foundations section 2.2.5 for logistic regression and 2.2.4.3 for UAR). In this context, the metric indicates the degree of overlap between labels regarding their affect ratings: High UAR values indicate that each affect rating is associated with a single label, while low UAR values indicate that each affect rating is associated with multiple labels.

- **Correlation of distance matrices:** I tested the correlation of stimuli distances between label-based distances and emotion-based distances. I determined distance matrices as explained in section 4.2.6.2 to calculate the Pearson correlations between them. High correlations indicate that stimuli that were rated as similar according to label ratings were also rated as similar according to affect ratings.

4.3 Results

The structure of the results section is as follows: Section 4.3.1 presents the participants perception of the rating items, as well as inter-rater reliability. Section 4.3.2 presents a detailed univariate analysis on the label item ratings, such as derivation of the salient label set. Section 4.3.3 presents the analysis of associations between rating items, e.g. correlations between affect dimensions as well as label ratings. Section 4.3.4 performs a cluster analysis of stimuli based on ratings to identify stimuli groups. Finally, section 4.3.5 presents the derived classification schemes that are based on the former results.

4.3.1 Rating items reliability and perception

Table 4.1 summarizes (1) raters perception of rating items, and (2) inter-rater reliability (IRR) of rating scales.

I summarized perception ratings through median and mean values across participants. Ordering rating items ascending by mean perception value, the ranking was valence $\mapsto$ label $\mapsto$ tense arousal $\mapsto$ energetic arousal. I tested the significance of difference in item perception through a pairwise Wilcoxon signed rank test with Bonferroni correction. The difference between valence and energetic arousal ($p < 0.05$) as well as tense arousal ($p < 0.01$) was significant. The difference between valence and label ratings was not significant ($p > 0.05$). The difference between label, energetic arousal and tense arousal was not significant (all $p > 0.9$).

I calculated IRR through intra-class-correlation (ICC) with a two-way random effects model for single raters as recommended by Koo and Li [81] (corresponding to an ICC(2,1) model). Calculation of IRR for label ratings was not possible due to the option of adding custom labels. Ordering rating items descending by mean IRR value, the ranking was valence $\mapsto$ tense arousal $\mapsto$ energetic arousal. Hallgren [54] recommends to interpret IRR values as *poor* for < 0.4, *fair* for 0.4 – 0.6, *good* for 0.6 – 0.75, and *excellent* for > 0.75. Consequently, reliability was high for valence, fair for tense arousal and poor for energetic arousal.

	perception		IRR (mean (95 % quantiles))	
	mean	median	agreement	consistency
label	2.9	3 (fair)	–	–
valence	2.2	2 (easy)	0.70 (0.64 – 0.76)	0.72 (0.66 – 0.77)
energetic arousal	3.6	3 (fair)	0.3 (0.24 – 0.38)	0.35 (0.29 – 0.43)
tense arousal	3.4	4 (hard)	0.44 (0.37 – 0.53)	0.49 (0.42 – 0.57)

Table 4.1: **Summary of rating item perception and reliability.** Perception rating range was 1 to 5, with 1 ≙ *very easy* and 5 ≙ *very hard*. IRR range was -1 to +1, with 0 ≙ reliability at chance and 1 ≙ perfect reliability.

4.3.2 Analysis of label ratings

4.3.2.1 Starting pool synonym pairs

43 % of participants reported *synonym-pairs* for starting labels, i.e. the starting pool to contain at least two synonymous labels. Of those, 90 % indicated that whimpering and whining where synonymous. I remapped the participant's respective labels to the more frequent one, which was *whining* in all cases. Other synonym pairs were *crying* and *screaming*, *moaning* and *screaming*, and *moaning* and *whining*, all of which were indicated exactly once.

4.3.2.2 Newly added labels

In total, 93 % of label ratings originated from the starting pool. The quantiles of number of newly added labels per participants were 0 (0 %), 0 (25 %), 1 (50 %), 2.5 (75 %), and 12 (100 %). Consequently, participants relied mostly on the starting pool labels.

The following labels were added at least twice (after spell checking): 7 × quietschen (squealing); 3 × hecheln (panting); 2 × erzählen (telling), gähnen (yawning), jauchzen (whooping), singen (singing), spielen (playing).

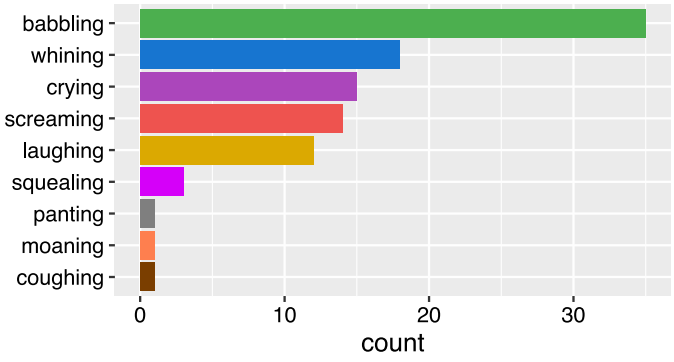

Figure 4.2: **Frequency of salient labels.** Labels are literal English translations from German.

4.3.2.3 Salient label set

I determined one consensus label per stimulus as the one most often voted, as the **salient label** (see section 4.2.6.3). Figure 4.2 shows the frequency of salient labels. This figure, as well remaining in the results section, displays literal translations from the original German labels as: schreien = screaming, weinen = crying, jammern = whining, husten = coughing, hecheln = panting, brabbeln = babbling, quietschen = squealing, and lachen = laughing. Two newly added labels *squealing* and *panting* prevailed, while all other labels originated from the starting pool. I highlight that the count of labels does not indicate real-world frequency of the respective call type, but merely the frequency in the data set.

Figure 4.3 shows a semantic map of the vocalization labels. The semantic map was constructed through non-metric multidimensional scaling (MDS) using label-based distances between stimuli (see section 4.2.6.2). MDS is a tool for projecting data points into a k dimensional space so that distances between data points correspond as close as possible to original distances. Unlike a metric MDS, a non-metric MDS interprets distances as rank orders rather than metric distances. The MDS implementation was provided by the R package `vegan v2.5-7` with function `metaMDS` with `monoMDS` engine, global regression mode, and rotation of axis according to the first two principal components. The stress was 0.1.

I highlight the following observations:

1. The greatest distance is between crying and babbling vocalizations. These are placed on opposing ends for the first principal component.
2. The arrangement of stimuli follows a "tilted horseshoe pattern". Stimuli are placed around the horseshoe in the following order: screaming $\mapsto$ crying $\mapsto$ whining $\mapsto$ breathing / moaning $\mapsto$ babbling $\mapsto$ squealing $\mapsto$ laughing. Coughing is placed as an outlier far away from all other vocalizations. Laughing and crying represent the open ends of the horseshoe.

4.3.3 Association between rating items

I aggregated participant votes for each stimulus as described in section 4.2.6.3. Figure 4.4 shows the relationship between aggregated affect and label ratings.

4.3.3.1 Association between affect scales

I highlight the following observations on the associations between affect scales:

1. The Pearson correlation between affect scales are: 0.66 for valence & energetic arousal, -0.84 between valence & tense arousal, and -0.33 for energetic & tense arousal. Consequently, valence and tense arousal have the highest association.

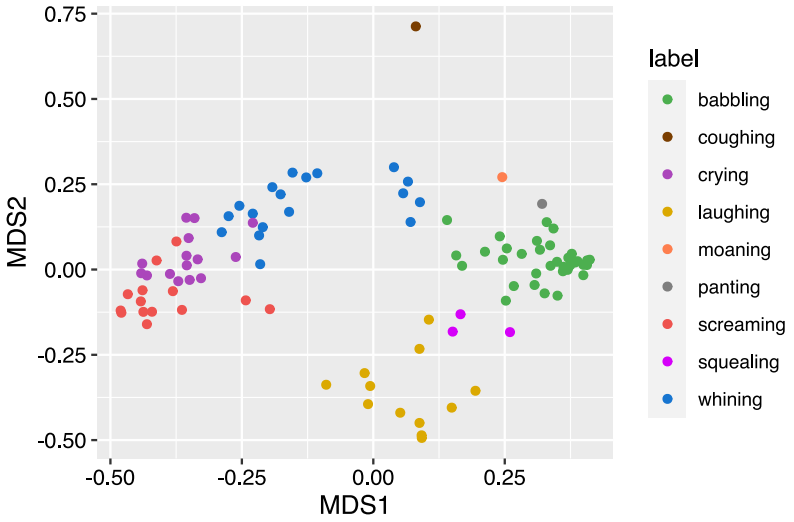

Figure 4.3: **Semantic map of salient labels**. Data points represent stimuli, color coded by their respective most voted label. Distances between stimuli represent their distances according to label ratings. Technically, this is a non-metric MDS of label-based stimuli distances.

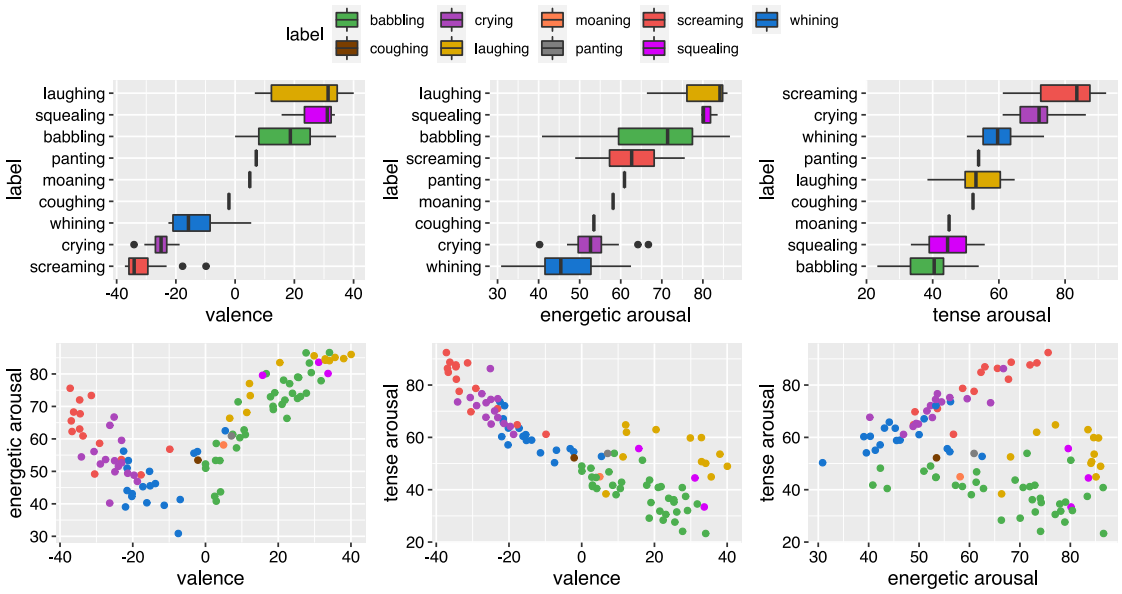

Figure 4.4: **Relationship between aggregated label and affect ratings.** Each data point represents a stimulus, color coded by its most salient label. A stimulus' affect rating is the average of all participants ratings. The top row shows single affect ratings vs label ratings, the bottom row shows dual affect ratings vs label ratings.

2. All affect scales are highly correlated inside the negative valence space. When subsetting stimuli to those with valence < 0, Pearson correlations are -0.57 for valence & energetic arousal, -0.89 for valence & tense arousal, and 0.81 for energetic & tense arousal.
3. When excluding stimuli with label *laughing*, the correlation between valence and tense arousal is -0.93. Consequently, both affect scales essentially represent the same concept, except for laughing vocalizations.

4.3.3.2 Association between label and affect ratings

Section 4.2.6.4 explained the procedure for calculation of the association between label and affect ratings. I calculated associations for all stimuli as well as for stimuli subsets based on aggregated valence ratings: Negative stimuli (average valence ratings -50 to -10), neutral stimuli (-10 to 10) and positive stimuli (10 to 50). Table 4.2 shows the resulting values.

I highlight the following observations:

1. Valence has the greatest overall association to label ratings. This is supported by the visualizations in Fig. 4.4, as well as the association values for all stimuli shown in Tab. 4.2. Tense arousal has the second strongest and energetic arousal the weakest association.
2. Screaming, crying and whining are highly associated to valence and tense arousal ratings. They might be viewed as a quantization of the negative valence space. Evidences for this are: (a) the arrangement of labels according to valence ratings correspond to their arrangement in the label-based MDS of Fig. 4.3, and (b) the high associations between label and valence / tense arousal ratings in the negative valence space shown in Tab. 4.2. However, the association is not perfect, i.e. emotion-wise there is some overlap between salient labels.
3. Babbling spans the greatest range in all affect dimensions among all labels, particularly for energetic arousal. It occupies the entire valence domain > 0, i.e. ranges from completely neutral to very positive.
4. Stimuli with neutral valence ratings have the smallest association to affect ratings, i.e. their separability by affect ratings is low. This applies particularly to *vegetative vocalizations* in the taxonomy of Buder et al. [21], comprising *panting*, *coughing* and *moaning*. They received nearly neutral ratings for all affect scales, but have 0 % overlap in label ratings. The respective association values shown in Tab. 4.2 support this.
5. Labels of stimuli with positive ratings are best differentiated by tense arousal, mainly because is has the highest association to laughing and babbling. Association values in Tab. 4.2 support this.

		all	neg.	neut.	pos.
aggregated ratings	valence	0.374	0.732	0.374	0.424
	energetic arousal	0.263	0.684	0.263	0.532
	tense arousal	0.346	0.735	0.346	0.622
distance correlations	valence	0.644	0.547	0.119	0.563
	energetic arousal	0.423	0.320	0.280	0.263
	tense arousal	0.526	0.447	0.250	0.563

Table 4.2: **Association between label and affect ratings**. Calculation methods are explained in section 4.2.6.4. *Aggregated ratings* shows the accuracy when predicting the most salient label based on aggregated affect ratings. Cell values indicate UAR values. *Distance correlation* shows the correlation between the label-based distance matrix and the respective affect distance matrix. Cell values indicate Pearson correlation values. Columns indicate whether all stimuli were used or limited to those with negative, neutral or positive ratings on average.

4.3.3.3 Including affect ratings into label MDS

I additionally investigated the influence of including distances for affect scales into MDS for producing semantic maps. The goal was to observe changes in stimuli placements and distances. Figure 4.5 shows the results for various combinations of rating items. I highlight the following observations:

1. The horseshoe shape observed in the purely label-based MDS (see Fig. 4.3) fundamentally remained even when combining label distances with any or all affect distance matrizes. The central difference to the purely label-based MDS is that neutral reflexive vocalizations *panting*, *coughing* and *moaning* mix into *babbling* and *whining* when affect ratings are considered. As previously explained, this is due to these stimuli being more similar in the affect domain.
2. Even when *excluding* label ratings from the MDS and just using all affect scales a similar horseshoe shape emerges (see plot 4 in Fig. 4.5). Again, the difference between *panting*, *coughing*, *moaning* and *babbling* is less pronounced in this case, and the distance between babbling and laughing is less pronounced as well.

4.3.4 Clustering of stimuli

In previous sections, I grouped stimuli by their most salient labels. As an alternative approach for stimuli grouping, I performed clustering on label distances. I tested the following clustering algorithms: partitioning around medoids (alias k-medoids) *PAM*, as well as agglomerative clustering variants *average*, *single*, *complete* and *ward* [136]. For each algorithm, I evaluated solutions for number of clusters $k \in \{2, \ldots, 7\}$.

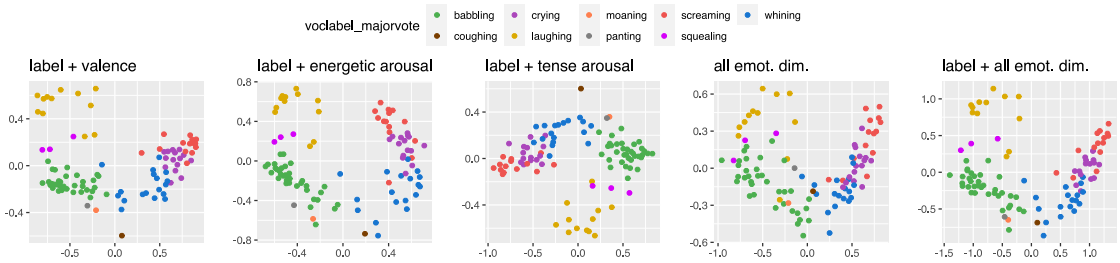

Figure 4.5: **Semantic maps for MDS based on combinations of various rating items.** All semantic maps were constructed through MDSs similar to Fig. 4.3. Titles of plots indicate the rating items included in the respective MDS.

For evaluation I used the average silhouette coefficient [141]. This metric indicates the performance of a clustering solution regarding cluster separability in the value range $[-1, 1]$, where 1 indicates perfect separability between clusters, 0 indicates no structure in data, and -1 indicates worse-than-random separability. I used the implementation of the R package `cluster v1.2.0` for clustering algorithms and silhouette calculation.

Figure 4.6 shows the silhouette scores for various cluster numbers and algorithms. I highlight the following observations:

1. Silhouette scores are generally low, with the highest silhouette score reaching 0.32 with PAM and 3 clusters. This indicates that stimuli separability based on labels is generally difficult.
2. The best clustering solutions were found for 2,3 and 5 clusters. Silhouette coefficients dropped for more than 5 clusters.

Figure 4.7 shows the clustering solutions with the highest silhouette score for each number of clusters. $k = 2$ differentiates between positive and negative vocalizations. $k = 3$ differentiates between laughing, positive non-laughing vocalizations and negative vocalizations. $k = 4$ separates coughing. $k = 5$ breaks negative vocalizations into slightly negative vocalizations (whining) and medium & strongly negative vocalizations (crying & screaming).

4.3.5 Derivation of classification schemes

Based on the previous findings, I proposed two classification schemes shown in Fig. 4.8. Both schemes are stylized semantic maps, i.e. they indicate classes in a spatial arrangement that indicates their relationship to each other [198].

I chose to primarily base these classification schemes on label and valence ratings: Label was the central rating item as it expressed linguistically expressed categories. Affect scales did not completely explain the variability between label ratings as shown in section 4.3.3.2, particularly regarding neutral reflexive vocalizations. I chose valence for supporting information as it was the affect scale with the highest inter-rater agreement as well as highest rater perception.

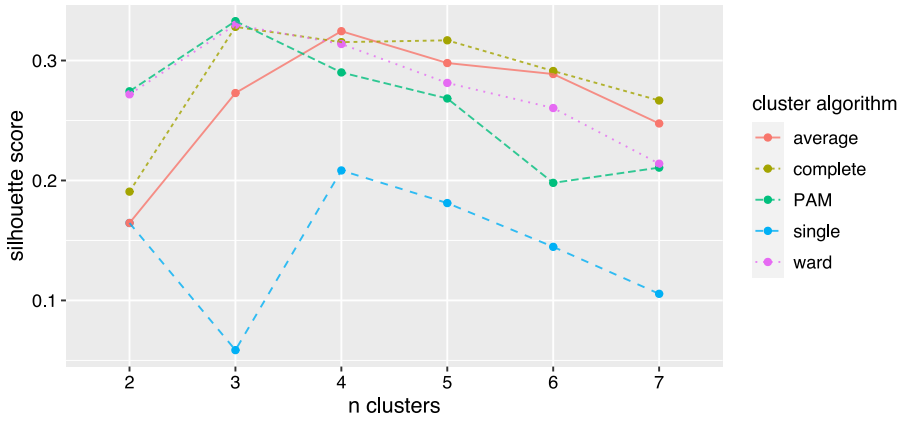

Figure 4.6: **Average silhouette scores for various clustering algorithms and cluster numbers.** All algorithms are agglomerative clustering variants, except for PAM.

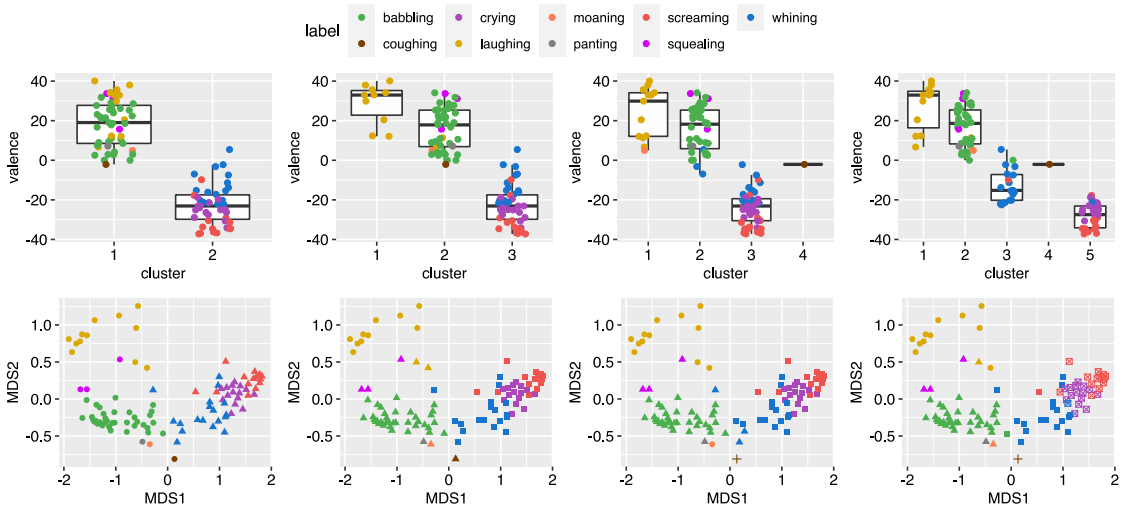

Figure 4.7: **Clustering solutions.** Each column of plots shows clustering solutions for number of clusters $k = \{2, 3, 4, 5\}$. For each k, the clustering was performed by the algorithm that reached the highest silhouette coefficient (see Fig. 4.6). The top row indicates valence ratings of cluster groups. The bottom row indicates groups through data points shapes in the MDS solutions based on valence + label distances.

The two schemes are as follows:

- **Horseshoe scheme**: Classes are based on the salient labels identified in section 4.3.2.3. However, I grouped neutral, voice-less reflexive vocalizations into a single class, comprising sounds such as *coughing, breathing, ...*. This represents the *vegetative sounds* group in the taxonomy of Buder et al. [21]. I did this to account for further of such vocalizations such as hiccup, sucking etc., all of which would share the property of involuntary, voice-less vocalizations with neutral affect ratings. The arrangement of classes is a stylized variant of the MDS projections shown in figures 4.3 and 4.5. Class placement particularly highlights similarity between classes based on their acoustic category, for example laughing and crying. The scheme additionally indicates the rough association to valence ranges as supporting information.
- **Mood scheme**: Classes correspond to the horseshoe scheme. However, the arrangement of classes indicates their expected valence ranges. Particularly classes screaming, crying and whining are indicated as a quantization of the negative valence space, according to the findings in section 4.3.3.2. Vegetative vocalizations are associated with neutral valence ratings. The remaining classes are merely roughly associated with neutral and positive valence ratings according to their greater valence-wise variance.

According to the cluster analysis in section 4.3.4, it is to be expected that classes are not perfectly separable. Separability might be increased by combining neighboring classes. For example, the "resolution" of the negative vocalizations might be increased by combining them into two or just one group. Also, babbling might be combined with either vegetative vocalizations or squealing to increase separability.

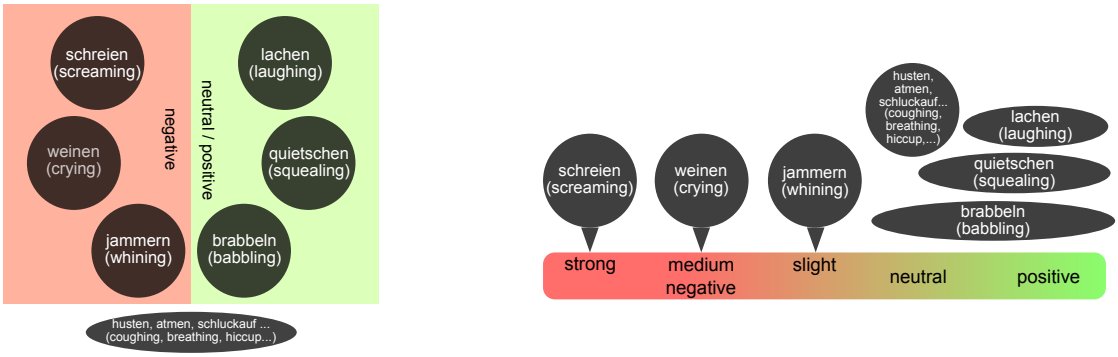

Figure 4.8: **Derived classification schemes** Left: Horseshoe scheme. Right: Mood scheme. Circles represent classes. The class *coughing, breathing,...* indicates a group for neutral vegetative vocalizations. Valence related information represents supporting information.

4.4 Discussion

One of the central results of this study is that *screaming*, *crying* and *fussing* might be viewed as a quantization of the negative valence space. Previous research reflects this: Barr [16, chapter 2] argued that crying is best understood as a graded signal. Pain scales likewise order classes of negative vocalizations as ordinal scales.

Regarding negative affective vocalizations, participants were likely biased by the starting pool of labels, preventing usage of alternate synonyms. For example, *crying* might have spanned a larger range if *whining* would have not been provided in the starting pool. Evidence for this is that almost all label ratings originated from the starting pool, particularly in the negative valence space. Consequently, future studies might research which labels provide an optimal quantization of the negative valence space, considering reliability as well as close distress association.

The final classification scheme shares similarities to the scheme of Buder et al. [21] (see state of research section 3.3.1). The class *laughing* is present in both; *Crying and fussing* corresponds to *screaming*, *whining* and *fussing*, i.e. my scheme differentiates between three instead of two distress vocalization classes; However, the entire *protophone* category with its numerous subclasses (vowels, grunts, gooing etc.) is contained entirely by the *babbling* class in my scheme. The only protophone subclass separated from *babbling* was *squealing*. This could indicate that laypersons indeed have a low capability to differentiate inside protophone-like vocalizations. However, this could also be an artifact of biasing through the starting pool label set. Possibly, *babbling* was to broad of a term and participants would have differentiated more if it was absent.

Oller et al. [124] introduced a concept named *functional flexibility*: This concepts means protophone vocalizations are flexible regarding the expressed emotion, while reflexive vocalizations are associated with fixed affects. The results of this thesis partially confirm this: Laughing, vegetative vocalizations, and whining/crying/screaming were strictly associated with positive, neutral, and negative valence. However, the *babbling* class, which closely corresponds with the protophone class, spanned the largest valence range, from completely neutral to positive. Consequently, it was indeed associated to a lesser degree with fixed valence values.

Another finding that is supported by my study is that separating vegetative vocalizations from protophones usually requires visual confirmation, to determine whether a vocalization was involuntary [21, 124]. There were various stimuli for which individual participants provided new labels associated with vegetative classes, such as *sucking* or *defication*, but which the great majority labeled as *babbling*. I hypothesize that *coughing* and *breathing* were recognized primarily because they are voice-less.

I highlight that study results might not be directly transferable to English. First and foremost, linguistically expressed categories are inherently influenced by language [12]. The literal translations of salient labels into English should be regarded as such: While *schreien* and *screaming* are literal translations, *crying* might semantically be the more appropriate term in the context of infant vocalizations, considering the frequency of usage in English research literature [15, 21, 164]. Similarly, *fussing* is primarily associated with

slightly negative vocalizations in these studies, however the literal translation *aufregen* is not commonly used in German. I hypothesize that *fussing* corresponds to *jammern*, which I translated with *whining*. Consequently, it is possible that replication of this study with native English speakers results in the discovery of different classes.

CHAPTER 5

Study (B): Comparison of Neural Network Types for Automatic Classification of Infant Vocalizations

5.1 Study goal

The goal of this study was to identify which neural network type among the currently most prevalent ones reaches the highest performance for infant vocalization classification. These types comprised convolutional neural networks, recurrent neural networks, fully-connected neural networks, as well as combinations of thereof. The precise research questions were:

- How does the arrangement of well-known network components affect the performance of network types?
- What is the best arrangement for each network type?
- Which type reaches the highest performance, given the best arrangement?

This study facilitates the understanding of representation learning systems with neural networks for automatic infant vocalization classification. There has not yet been an in-depth study comparing various network architectures while considering the influence of their configuration. The network architecture is the core of any representation learning system and thus of particular importance. Previous studies either directly adapted computer vision architectures, such as AlexNet [189], or presented a custom architecture without further justifications for the chosen configuration [23, 87, 173]. This relates to the research gaps discussed in section 3.3.2.

All networks operated on Mel-scaled spectrogram inputs, as this currently is the most prevalent audio representation in deep learning systems. The task was a monophone classification task for infant vocalizations into classes *neutral*, *fussing*, and *crying*. I defined an architecture scheme to derive all network types in an uniform manner. For each type, I performed a semi-random hyperparameter search. I employed regression trees to identify the most auspicious hyperparameter ranges and subsequently focused the search spaces. I finally selected the best configurations of each type and compared their performances in a contest-like setup.

Study (C) builds on the results of this study. The results of this study showed that CNNs reached the highest performance. Study (C) performs a more detailed investigation of CNNs, applied to another data set.

5.2 Materials and methods

The structure of the methods section is as follows: Section 5.2.1 presents the acoustic data set. Section 5.2.2 presents the general classification pipeline including the neural network. Section 5.2.3 presents the neural network architecture scheme, which was the basis for defining the various ntwork types. Section 5.2.4 presents the hyperparameter search space for each network type. Sections 5.2.5 and 5.2.6 present the training and evaluation setup, respectively. Finally, section 5.2.7 presents the experimental setup, i.e. the manner in which the investigation on network types and configurations were executed.

5.2.1 Acoustic data set

The data set for this study was taken from the ComParE 2018 challenge. This data set was already briefly introduced in section 3.3.2. Marschik et al. [101] provided the data set, which originates from their study on early detection of neurodevelopmental disorders. They named it the *Cry Recognition In Early Development (CRIED)* data set. It comprises 20 infants (10 male, 10 female) recorded in various sessions across the first months of life. The segmentation criterion for unit extraction was the breath group criterion, which resulted in 5587 units (see foundations section 2.1.2 on this criterion). Units were classified as either *crying*, *fussing* or *neutral* [150].

Figure 5.1 shows the duration histogram of the resulting units. The distribution of durations is considerably positively skewed. The class distribution is unbalanced as well, with the classes *neutral*, *fussing* and *crying* accounting for 79.9 %, 14.5 % and 5.6 % of data set examples, respectively.

The coding was performed by two trained experts based on audio-video data, i.e. the coders could see an infant's face while assigning classes. The procedure ensured agreement between both raters. However, some degree of information necessary for class separability might have been lost when discarding videos, as other studies reported that visual information influences vocalization coding [116, 124, 163].

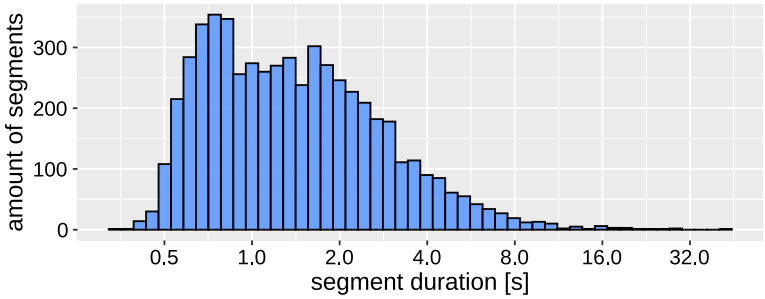

Figure 5.1: **Histogram of segment durations** of the data set presented by Marschik et al. [101], Schuller et al. [150]. The x-axis is $\log_2$-scaled to provide better visibility of the positively skewed distribution. The quartiles are at $Q_0 = 0.33\,\text{s}$, $Q_{25} = 0.8\,\text{s}$, $Q_{50} = 1.3\,\text{s}$, $Q_{75} = 2.1\,\text{s}$ and $Q_{100} = 41.1\,\text{s}$.

5.2.2 Classification pipeline surrounding the neural network

This section describes the processing pipeline surrounding the neural network, which performs the core of the prediction. I modeled the task as a monophone classification task. Target vectors use one-hot encoding for network training and integer encoding for evaluation, with three classes corresponding to *neutral*, *fussing* and *crying* (see foundations sections 2.4.1 and 2.2.3 on these terms).

The pipeline fundamentally follows the blueprint presented in section 2.4.3. Fig. 5.2 visualized the pipeline. The implementation of the blueprint follows a common pipeline in modern deep learning classification algorithms [176, chapter 2][60, 108]. The process is as follows:

1. **Preprocessing:** Audio segments were first converted to mono and subsampled to 16.000 Hz, which is common in CP [150, 177]. Each segment was normalized to an absolute peak amplitude of 1 to ensure similar volume levels across all audio segments.
2. **Feature map conversion:** The feature map type was the log Mel-scaled spectrogram as the audio representation, as defined in section 2.1.1. This currently is the most prevalent feature map choice in modern deep learning systems (see state of research section 3.2).

   The implementation of the log Mel-scaled spectrogram roughly followed the procedure of the DCASE 2018 task 1 baseline system [110]: First, signals were transformed into the magnitude spectrogram as defined in Eq. 2.6, were then transformed into Mel-scaled spectrograms as defined in Eq. 2.9, and finally log-transformed as defined in equation 2.7. The Mel-scale and filterbank implementation followed the proposal of Slaney [158].

   The exact parameters of the feature maps were as follows: The window function was the hanning window, window length was 30 ms, the hop size $h = 15\,\text{ms}$, the

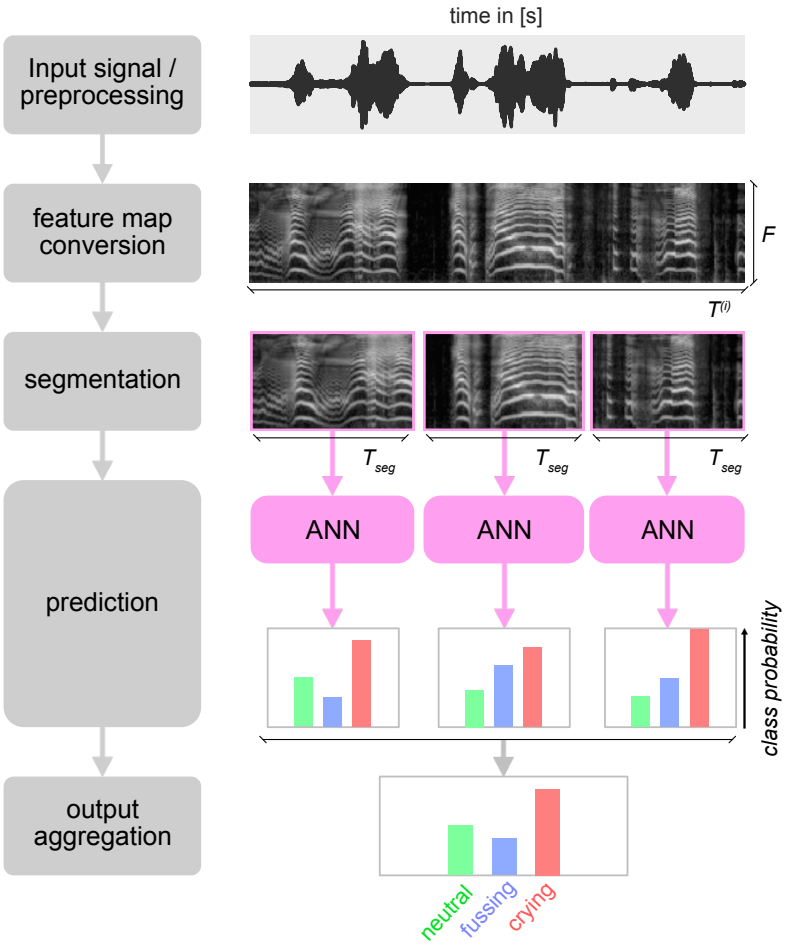

Figure 5.2: **Overview of the classification procedure.**

FFT size of $N = 512$ and $m = 80$ Mel bands. Spectrograms were z-standardized with global statistics calculated on the training set. The feature extraction process was implemented through the python library `librosa v.0.7.1` [103].

The resulting feature map for signal (i) is a two dimensional matrix $\boldsymbol{M}^{(i)} \in \mathbb{R}^{T^{(i)} \times F}$, where $T^{(i)}$ is the time axis and F is the frequency axis.

3. **Segmentation:** Mini-batch training of neural networks (see foundations section 2.3.3.2) requires all input segments to have equally sized dimensions, even if the model itself aggregates across the time axis. Therefore, I implemented feature map segmentation to bring all feature maps to the same temporal length T_{seg}. Signals with $T_{\text{seg}} > T^{(i)}$ were zero-padded at the beginning, following Wagner et al. [177]. Signals with $T_{\text{seg}} < T^{(i)}$ were segmented into non-overlapping windows, i.e. the segment size was equal to the hopsize following Hershey et al. [60]. The last segment was allowed to overlap with the penultimate when $T_{\text{seg}} \bmod T^{(i)} \neq 0$.

   The challenge in choosing T_{seg} was the heavy positive skew in signal durations

in the data set (see Fig. 5.1). I chose $T_{\text{seg}} = 5\,\text{s}$, so that most segments (95 %) are not truncated to ensure that little information is discarded. Increasing T_{seg} further had diminishing returns due to the positive skew in segment durations (see foundations section 2.4.3 on the tradeoff when choosing T_{seg}).

After segmentation, each feature map $\boldsymbol{M}^{(i)}$ is converted into a list $\{\dot{\boldsymbol{M}}^{(i,s)}\}_{s=1,\dots,S^{(i)}}$, where $\dot{\boldsymbol{M}}^{(i,s)} \in \mathbb{R}^{T_{\text{seg}} \times F}$ and $S^{(i)} \in \mathbb{N}$ is the respective number of segments. For training, segments inherited the labels of their parent signals.

4. **Prediction:** The neural network predicts each segment $f(\dot{\boldsymbol{M}}^{(i,s)}) = \hat{\boldsymbol{p}}^{(i,s)}$, where $\hat{\boldsymbol{p}}^{(i,s)} \in [0,1]^3$ indicates class probabilities.
5. **Output aggregation and binarization:** To derive the final integer-encoded class label $\hat{y}^{(i)}$ per signal, I averaged network output scores across all of its respective segments and chose the most probable class:

$$\hat{y}^{(i)} = \text{argmax}\left(\frac{1}{S^{(i)}} \sum_{s=1}^{S^{(i)}} \hat{\boldsymbol{p}}^{(i,s)}\right) \tag{5.1}$$

5.2.3 Neural network architecture scheme

All neural network types evaluated in this study were derived from a unified parent architecture scheme. The scheme was based on the *CLDNN* architecture proposed by Sainath et al. [143]. Figure 5.3 shows the scheme. It defines a cascade of **stages** that collectively define the model, which progressively process the incoming feature map to a vector of class probability scores.

For further explanations on the layers and components referred to in this section, please refer to section 2.3 and particularly section 2.3.2.

Let $\boldsymbol{M}^{T \times F}$ be an input spectrogram. The input layer expands the dimensionality of the input spectrogram by adding an empty channel dimension $\mathbb{R}^{T \times F} \mapsto \mathbb{R}^{T \times F \times C=1}$. The output layer contains three fully-connected neurons with softmax activation functions that output class probabilities.

Both frequency and time aggregation stages are mandatory stages that tensors have to pass through in the indicated order. They progressively reduce the tensor dimensionality to finally produce a one-dimensional tensor to be fed into the output layer. The respective removed dimension is integrated into the channel dimension, although the actual aggregation operations depend on the stage's parametrization.

The convolutional, recurrent and fully-connected stages contain a stack of layers of their respective type, e.g. the convolutional stage contains convolutional layers. These three processing stages are optional, i.e. each one can be skipped. This allows for a flexible configuration of neural networks by switching stages in and out and is essential for their systematic comparison in this study: I defined **Network Types** through the processing stages that are included in the network. For example, the network type C-FC-NN contains a **c**onvolutional and a **f**ully-**c**onnected stage and skips the recurrent

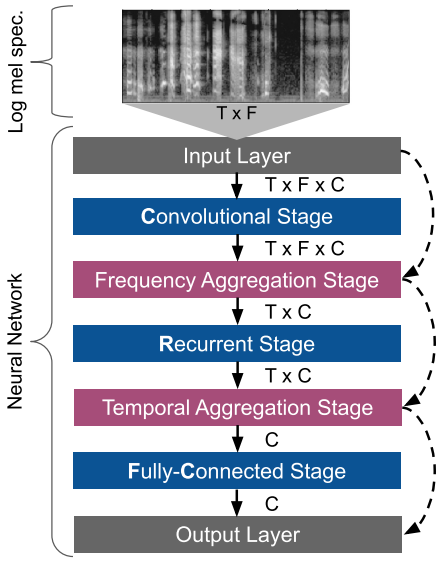

Figure 5.3: **Parent neural network architecture scheme.** Each box represents a network stage. Blue boxes represent processing stages, pink boxes represent aggregation stages. Arrows indicate connection pathways between stages. Dotted arrows indicate connections between aggregation stages that skip processing stages. Indications between stages denote tensor dimensionalities, with axis meanings T = *time*, F = *frequency*, and C = *channel*. For example, the temporal aggregation stage inputs a two-dimensional tensor with axes *time* × *channel* and outputs an one-dimensional tensor.

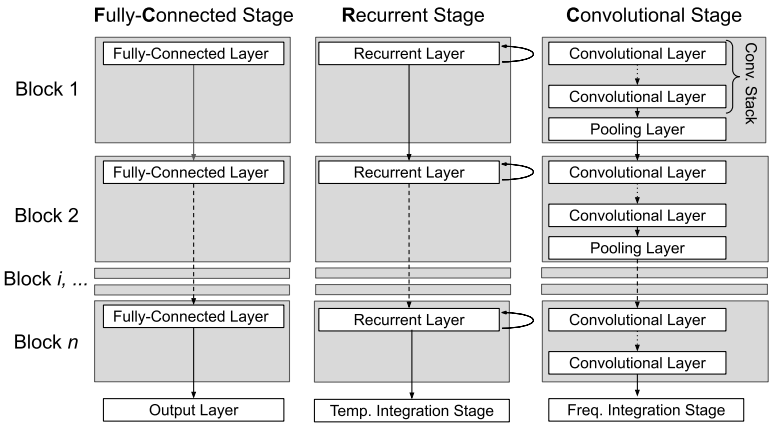

Figure 5.4: **Schemes of the processing stages.** Details are provided in the text.

stage. Consequently, the scheme is a general template for the current popular network types (see foundations section 2.3.4 on examples)[1].

Each stage constrains the dimensionality of its input tensor as indicated in figure 5.3. In contrast to the aggregation stages, the three processing stages do not alter a tensor's dimensionality. The proposed order of stages directly results from the tensor dimensionality progression as well as from their hypothesized functions: The convolutional stage is hypothesized to produce a feature map by extracting local features from spectrogram patches; the recurrent stage processes local features at each time step while incorporating temporal information; and the fully-connected stage processes a set of static, summative features [22, 46, 143].

While I adapted the basic arrangement of skippable convolutional, recurrent and FCLs from Sainath et al. [143], my scheme differs as follows: (1) The reference architecture defines layer stacks with fixed configurations that I generalized to *stages*, e.g. my *recurrent stage* represents the generalized case of the original *LSTM layers*. (2) I added the temporal aggregation stage as this studies' task requires one-dimensional outputs; the reference architecture was developed for speech recognition, which requires time-distributed outputs.

Figure 5.4 shows the architecture schemes for the processing stages. Each stage contains a variable number of consecutive blocks. For fully-connected and recurrent stages, each block merely contains a layer of the respective type as in the reference architecture [143]. A convolutional block however contains a stack of convolutional layers followed by a pooling layer, which is the most common architecture principle for CNNs (see foundations section 2.3.4 on examples). The last convolutional block omits the pooling layer to avoid applying two pooling layers in succession through the subsequent aggregation stage.

5.2.4 Hyperparameter search space

A network's actual performance depends on the parametrization of its layers. The objective of the hyperparameter search is to find the configuration that maximizes a network's performance. Section 5.2.4.1 presents the network hyperparameters and section 5.2.4.2 the search space.

5.2.4.1 Network hyperparameter definitions

Theoretically each hyperparameter in each network's layer could be chosen individually in the search process, e.g. each layer could contain a different activation function. The downside to this approach is that it induces codependencies into the parameter selection process, e.g. choosing a layer's activation function depends on the priorly chosen amount of network layers. Consequently, networks would contain different amounts of

[1]Note that network naming is more specific than usually is the case. Usually, *CNN* is an umbrella term for networks containing convolutional layers in general, regardless of whether they contain FCLs as well. However, in this study a *C-NN* specifically means that a network contains a convolutional stage and no FCLs

Network stage	Hyperparam. identifier and domain	Explanation
convolutional, recurrent, and fully-connected stage	number of blocks $\in \mathbb{N}$	Number of consecutive blocks in the stage, the "stage depth".
	number of units in the first layer $\in \mathbb{R}$	Number of units (alias *neurons* or *kernels*) in the first layer of the stage relative to the absolute number of units in the last layer of the preceding stage. For example, a value of 2 means that the first layer of this stage has twice as much units as the last layer of the preceding stage. If this is the first stage, the value directly indicates the absolute amount of units, i.e. the reference value is 1.
	unit growth $\in \mathbb{R}$	Factor by which the the number of units grows with each block. For convolutional stages, all convolutional layers in a stack contain the same amount of units.
	batch normalization $\in \mathbb{B}$	Indicates whether batch normalization is applied after each stage layer (before the activation function).
	activation $\in f : \mathbb{R} \mapsto \mathbb{R}$	Type of activation function applied to all layers inside the stage.
convolutional stage	kernel shape $\in \mathbb{R}^2$	Kernel size of all convolutional layers.
	conv. type $\in f : \mathbb{R}^{\mathbb{N}^3} \mapsto \mathbb{R}$	Type of convolution operation performed by each convolutional layer, e.g. conventional convolution or residual convolution.
	stack size $\in \mathbb{R}^2$	Number of consecutive convolutional layers inside a block, see Fig. 5.4.
	pooling shape $\in \mathbb{R}^2$	Pooling size and stride of all pooling layers.
	pooling type $\in f : \mathbb{R}^{\mathbb{N}^2} \mapsto \mathbb{R}$	Type of pooling operation performed by all pooling layers.
recurrent stage	recurrent type	Recurrent unit type used, e.g. LSTMs or GRUs.
	bidirectional $\in \mathbb{B}$	Whether each recurrent layer is applied unidirectional or bidirecitonal. For bidirectional connections, half of the layer units run forward and half backward.
temp. and freq. aggregation stage	operation $f : \mathbb{R}^{N_1 \times N_2} \mapsto \mathbb{R}^{N_3}$	Operation to reduce tensor dimensionality. The frequency integrator applies a time-distributed operation, i.e. the same operation at each time step.

Table 5.1: **Stage Hyperparameters.** All hyperparameters are stage-specific even if stages share the same parameter identifier. For example, each aggregation stage contains its own hyperparamter *operation* rather than sharing the same operation.

hyperparameters depending on their configuration even if they belong to the same network type. This prevents the application of ordinary statistical methods to analyze the relationship between a network's configuration and its performance, e.g. we could not simply regress from the activation function onto the performance. Additionally, the search space becomes unmanageably large as each parameter in each individual layer adds another dimension to the hyperparameter search space.

To circumvent these problems, I defined a fixed set of hyperparameters for each stage whose values can be chosen independently of one another (see Tab. 5.1). The stage's layers are then configured according to these parameters. This ensures the same number of hyperparameters per network type and the applicability of ordinary statistical methods.

As shown in Tab. 5.1, most stage hyperparameters merely constrain contained layers to share certain settings, e.g. all stage's layers share the same activation function. However, layers usually don't share the same number of units (alias *neurons* or *kernels*) inside a stage, but successively grow or shrink with each layer (see foundations section 2.3.4 for examples) [46, 55, 155, 194]. The number of units $U^{(s,b)} \in \mathbb{N}$ is calculated with respect to the number in the preceding layer as

$$U^{(s,b)} = \begin{cases} U_{\text{first}}^{(0)} & \text{if } s = b = 0 \\ U_{\text{first}}^{(s)} \cdot U^{(s-1,B^{(s-1)})} & \text{if } s > 0 \wedge b = 0 \\ U^{(s,b-1)} \cdot G^{(s)} & \text{else} \end{cases}, \tag{5.2}$$

where $s \in \{0, 1, 2\}$ is the stage index, $b \in \{0, \ldots, B^{(s)-1}\}$ is the block index inside the stage, B_s is the *number of blocks* hyperparameter, $U_{\text{first}}^{(s)}$ is the *number of units in the first layer* parameter, and $G^{(s)}$ is the *unit growth* parameter of stage s.

I designed the stage hyperparameters and the rules by which their layers are constructed to reflect prevalent network design principles found in previously published networks, e.g. applying the same activation function across the entire network, applying the same recurrent or convolutional type across the entire network, ensuring smooth unit growth across layers etc. (see foundations section 2.3.4 for network examples) [26, 36, 42, 56, 60, 84, 110, 153, 169, 177].

The drawback of this approach is that certain network configurations were excluded from the search space a priori, which theoretically are valid according to the parent architecture scheme, such as CNNs with varying kernel shapes across the network. However, this drawback did not inhibit a fair comparison between network types, as their stages were subjected to the same restrictions regardless of their application to a network type, which ultimately was the focus of this research.

5.2.4.2 Network hyperparameter ranges

Among the set of possible networks types resulting from the network scheme, I chose to evaluate R-NNs, R-FC-NNs, C-NNs, C-FC-NNs, and C-R-NNs. I omitted FC-NNs and C-R-FC-NNs to reduce computation time: Pure fully-connected networks are unable to make use of the grid-structured arrangement of the input and have already been proven

Stage	Parameter	Network Type				
		R-NN	R-FC-NN	C-NN	C-FC-NN	C-R-NN
convol. stage	activation	–	–	ReLU, ELU, LeakyReLU	←	←
	n. of blocks	–	–	2, 3, 4	←	←
	n. units first l.	–	–	8, 16, 32, 64	←	←
	unit growth	–	–	1, 2	←	←
	batch norm.	–	–	Yes, No	←	←
	kernel shape	–	–	(3,3), (1,5)	←	←
	pooling shape	–	–	(2,2), (1,2)	←	←
	conv. type	–	–	Plain, Residual	←	←
	pooling type	–	–	Max, Average, Stride	←	←
	stack size	–	–	1, 2, 3	←	←
freq. agg.	operation	Flattening	←	GAP, GMP, Flattening	←	←
recurr. stage	activation	Tanh	←	–	–	←
	n. of blocks	1, 2, 3, 4	←	–	–	1, 2
	n. units first l.	8, 16, 32, 64, 128	←	–	–	0.5, 1, 2
	unit growth	0.5,1,2	←	–	–	1
	batch norm.	No	←	–	–	←
	recur. type	GRU, LSTM	←	–	–	←
	bi-direcitonal	Yes, No	←	–	–	←
temp. agg.	operation	GAP, GMP, Attention, Last Step	←	GAP, GMP, Attention, Flattening	←	GAP, GMP, Attention, Last Step
fully-conn. stage	activation	–	–	–	←	–
	n. of blocks	–	–	–	←	–
	n. units first l.	–	–	–	←	–
	unit growth	–	–	–	←	–
	batch norm.	–	–	–	←	–

Table 5.2: **Search spaces of all network types.** The two leftmost columns indicate the network stages and parameters as defined in Tab. 5.1. Cells indicate parameter ranges, depending on the network type. A "-" represents empty ranges for skipped stages. An arrow ← indicates that the respective range corresponds to the one from the neighboring left cell.

to be outperformed by networks with weight sharing (CNNs and RNNs) in the DCASE competition [108, 110]. C-R-FC-NNs are computationally most expensive to train while also having the largest parameter search space; I expected the results on C-R-NNs and R-FC-NNs to sufficiently translate to this type.

Table 5.2 presents the parameter ranges for all network types. Justifications and further explanations for the presented ranges are as follows. Please refer to section 2.3.2 for further details on the layer types and to section 5.2.3 for an introduction on the referenced networks LeNet[90], AlexNet[85], VGGNet[155] and ResNet[55].

Convolutional Stage: The convolutional type *plain* means that each schematic convolutional layer in figure 5.4 directly corresponds to a simple convolutional layer and the *residual* type means that each schematic convolutional layer in figure 5.4 employs an ResNet like *residual block*. However, all layers in the first convolutional block always consist of *plain* type layers in correspondence to ResNet[2]. I included these types as they currently are the most prevalent CNNs in the DCASE and CP contests [4, 26, 56, 66, 73, 84, 96, 110, 153, 169, 170, 180, 188].

In view of the relatively small data set, I set the ranges for the parameters *depth*, *number of units in the first layer*, *stack size* and *unit growth* so to scan for smaller networks as well. The values for the quadratic kernel and pooling shapes are in accordance to the standard design pattern in CNNs for computer vision. I additional included one-dimensional kernel and pooling shapes across the frequency axis as these are more prevalent in networks for audio detection and speech recognition that preserve time frame alignment, such as CRNNs [22, 92, 143]. I included LeakyReLU [97] and ELU [29] as parameter-free alternatives of the commonly used ReLU activation. Max, average and stride pooling were adapted from LeNet, VGGNet and ResNet.

Recurrent Stage: Gated recurrent units (GRU) and long short-term memory units (LSTM) [27] are the two most common recurrent types in CP [47, 67, 166, 177, 197]. Unidirectional [177, 197] as well as bidirectional [67, 166] networks are popular. I used the CuDNN implementations[3] to reduce the training time of the recurrent units. However, as these implementations do not allow to alter the activation function nor implement recurrent batch normalization [31] I fixed the corresponding parameter ranges to the implementations' preset values.

Frequency aggregation stage: R-NNs and R-FC-NNs employed flattening aggregation exclusively to directly convert frequencies to channels without information loss. For networks with convolutional stages, flattening and 1D GAP correspond to the vertical aggregation operations of VGG and ResNet, respectively. I additionally tested 1D GMP, as this is the most common intermediate pooling operation inside convolutional stages and consequently could prove advantageous as a global aggregation operation.

Temporal aggregation stage: The temporal aggregation operations were adapted from Mirsamadi et al. [113]. Particularly attention pooling was widely employed in the 2018 ComParE challenge [47, 166]. All pooling types incorporate outputs at every time

[2]Note that, while the original residual block definition also used batch normalization layers, stride-pooling and ReLU-activations, these parameters are subject to the other hyperparameters in this study.

[3]`https://developer.nvidia.com/cudnn`

step, while *last step* means that only the output of the last time step is forwarded to the next layer (alias *many-to-one*-prediction). As for convolutional stages last-step-aggregation is not applicable as units do not carry internal states, I used flattening corresponding to the vertical aggregation operation of more conventional CNNs.

5.2.5 Training setup

The optimizer algorithm was Adam with standard parameters [78]. As the loss I chose weighted CCE in accordance with Wagner et al. [177], where I calculated weights separately for the training and validation set. I applied early stopping with a patience of 20 epochs. Training examples were shuffled between each epoch, the batch size was 32. The training and evaluation procedure was implemented in python through the deep learning framework `keras v.2.3.1` with backend `tensorflow-gpu v1.14.0` and `CUDA v. 10.0.130` (see foundations section 2.3.3 for definitions of these terms).

5.2.6 Evaluation setup and performance metric

The ComParE 2018 challenge setup specified a data split with 50 % of infants being used for development and testing, respectively [150]. The challenge recommended leave-one-subject-out-cross-validation (LOSO) for hyperparameter tuning. LOSO is a special kind of cross-validation, where folds are separated by subjects to avoid the *album effect* (see foundations sections 2.4.4 and 2.2.4.6 for these concepts).

As cross-validation is uncommon in deep learning [46], I adapted the alternative approach of the deep learning baseline systems [150, 177], which used a fixed training-validation-test split. I assigned 3 infants from the development set as a fixed validation set. I selected infants to produce similar class distributions in the training and validation set. The final data split used 35 %, 15 % and 50 % of subjects for training, validation and testing, respectively.

I measured validation set performance directly through the validation loss. I measured the test set performance through UAR metric in accordance to the ComParE challenge setup [150] (see foundations section 2.2.4.3 for this metric). Due to random layer initializations and random example shuffling between epochs, validation and test performance could vary between runs of the same configuration. For this reason, I trained and evaluated each network configuration thrice and averaged results.

5.2.7 Experimental setup

The basic notion was to design the comparison between network types by the model of competitions such as the DCASE or ComParE: First, a hyperparameter search is conducted for each network type to evaluate configurations on the validation set. Then, each type's most auspicious configurations are selected as competitors to be evaluated on the test set. The configuration with the highest test performance prevails, representative of its respective network type.

However, the parameter space presented in section 5.2.4.2 was too large for an exhaustive grid search. Instead I conducted a random search, divided into a coarse and a fine search phase (see foundations section 2.2.2.3 on these terms). The details of the procedure were as follows:

1. **Coarse Search and Analysis:** I generated a random sample of 500 configurations per type drawn from the search space described in section 5.2.4.2. Each configuration was trained and evaluated as described in section 5.2.6. To identify the most auspicious parameter settings per network type, I performed a regression analysis from the validation performance on the network parameters through regression trees (see explanation below).
2. **Fine Search and Analysis:** I focused the search space for each type according to the results of the prior regression analysis. I generated another random sample of 500 configurations per type from the restricted parameter space. I conducted another regression tree analysis on the results to gain additional insights on parameter importance. As this regression analysis did not influence further parameter tuning, I regressed directly from the test performance on the parameters.
3. **Comparison between types:** I selected n models for each type from the fine search sample ranked by validation performance to be compared regarding their test set performance.

I employed regression trees to model the relationship between performance P and the hyperparameters $\boldsymbol{\theta}$. For this, I set $\boldsymbol{\theta}$ as the features, P as the target, and build the regression tree. Each model's tree path leading to the leaf with the *highest* target value indicates the combination of hyperparameter values that resulted in the best performance on average. I utilized this fact to identify the corresponding optimal parameter setting for each network type and focus the fine search space accordingly. Other regression algorithms might have been used in a similar manner, e.g. multiple linear regression. However, regression trees have various advantages for this application, such as (1) ability to operate on mixed-type features, (2) account for feature interactions (see foundations section 2.2.5.2 for further explanations on regression trees).

This search strategy can be viewed as random search, interleaved with an exploitation step. Other more sophisticated hyperparameter search strategies might have been used instead of my approach, such as Bayesian Optimization. The reasons for favoring my approach were as follows: (1.) Simplicity of random search, as discussed in section 2.2.2.3. (2.) Validation set overfitting: Hyperparameters are tuned on the validation set, which carries the risk of overfitting the validation data while reducing test performance. The validation set in this study was unavoidably small due to the ComParE challenge's original 50/50 development/test split. Consequently, a mismatch between validation and test set was to be expected a priori. This mismatch has already been documented implicitly by the 2018 ComParE challenge [150], where validation and test performances of baseline systems were considerably decorrelated. By favoring exploration over exploitation in the search process, I reduced the risk of overfitting the validation data. (3.)

Assessment of hyperparameter importance: Rather than merely finding good network configurations, the goal was to assess hyperparameter importance as this carries greater generalizable knowledge gain. Randomly drawn samples facilitate the application of statistical analysis on hyperparameter importance.

5.3 Results

The structure of the results section corresponds to the procedure outlined in section 5.2.7. Sections 5.3.1 and 5.3.2 present the analysis on the fine and coarse hyperparameter search results, respectively. Section 5.3.3 presents the comparison between network types.

5.3.1 Coarse search analysis

Subject of the analysis in this section was the random sample of 500 configurations per network type drawn from the initial search space described in Tab. 5.2. I ensured a balanced distribution of hyperparameter values for each network type. I fitted one regression tree per network type, with the predictor variables being the hyperparameters and the target variable being the validation loss (averaged over three runs, respectively, see section 5.2.6).

Regression trees were implemented with the `R` package `rpart v.4.1-15` [171]. This implementation optimizes the mean squared error metric. As this is prone to outliers, the target variable was preprocessed to approximate normal distribution using the ordered quantile technique as implemented by the `R` package `bestNormalize v.1.4.2`. Each tree was pruned to the depth that yielded the lowest error in 10-fold cross-validation. All regression tree hyperparameters were set to default values (see section 2.2.5.2 for further explanations on regression trees).

All complete regression trees are displayed in Fig. 9.1. For each tree, I extracted the tree path from the root to the leaf with the highest target value, as the corresponding branches indicate each type's optimal hyperparameter combination. The left column of Tab. 5.3 summarizes these **best performing tree paths**.

The importance of hyperparameters for raising performance is indicated twofold:

1. Splitting rules that occur higher up in the hierarchy, i.e. first in a regression tree rule chain, are of higher importance. This is due to the greedy approach in decision tree building. [69].
2. Each splitting rule indicates the coefficient of determination R^2. This value indicates the gain in R^2 to the fit produced by the respective split, i.e. the proportion of explained variance in the target variable [171]. Consequently, it can be viewed as a measure of the statistical effect size of the splitting rule, with $R^2 > 0.04, > 0.25$ and > 0.64 indicating low, medium and high effect sizes, respectively [168].

I highlight the following observations on the best performing tree paths:

1. All best performing tree paths include all aggregation operations present in the respective network type. The respective splitting rules have at least medium effect sizes R^2, except for C-R-NNs with small effect sizes. This emphasizes that aggregation operations are the overall most influential hyperparameters.
2. All best performing tree paths exclusively contain 1D GAP for temporal aggregation.
3. The frequency aggregation operation *flatten* always reduced performance for network types with convolutional stages.
4. Batch normalization always reduced performance when applied to FCLs.

5.3.2 Fine search analysis

For the fine search process, the initial search space presented in Tab. 5.2 was restricted according to the best performing tree paths of the coarse search shown in the left column of Tab. 5.3. I drew another random sample of 500 configurations per network type from the restricted search space for training and evaluation. For R-NNs only 72 configurations remained, which allowed for an exhaustive grid search.

Corresponding to the coarse search analysis, I fitted one regression tree per network type on the results, however on the test UAR. The complete trees are displayed in Fig. 9.2. The right column of Tab. 5.3 shows the resulting best performing tree paths. It can be seen that the overall effect sizes of splitting rules R^2 were lower compared to the coarse search, i.e. the impact of hyperparameter choices decreased in general.

I highlight the following observations:

1. All best performing tree paths contain rules for limiting the number of blocks. For convolutional stages with larger amounts of blocks, limiting the stack size alternatively increases performance, as shown by the complete regression trees of C-NNs, C-FC-NNs and C-R-NNs (see Fig. 9.1 and 9.2). This indicates that increasing total network depth generally correlates with reduced performance. R-NNs are an exception by not indicating a rule for limiting depth; however, these networks were the shallowest to begin with.
2. Ultimately, all networks with convolutional stages exclusively chose 1D GMP for frequency aggregation, as indicated by the best performing tree paths of the coarse or fine search.
3. Ultimately, all networks with recurrent stages favored GRU-type units over LSTM-type units, as indicated by the best performing tree paths of either the coarse or fine search.

5.3.3 Comparison of network type performances

5.3.3.1 Comparison of overall test performances

Figure 5.5 gives an overview of the configuration's test performances of the fine search sample, i.e. 500 configurations per network type (72 for R-NNs). Networks with con-

volutional stages reached a maximum performance of ≈ 75 % and recurrent networks (i.e. those without convolutional stages) reached ≈ 73 %. Fully-connected stages did not increase the performance compared to the corresponding network type without a fully-connected stage, neither on average nor for the maximum performance (R-NNs vs. R-FC-NNs, and C-NNs vs. C-FC-NNs). The hyperparameter setting of each type's best configuration (marked with a red asterisk in Fig. 5.5) is stated in Tab. 9.1.

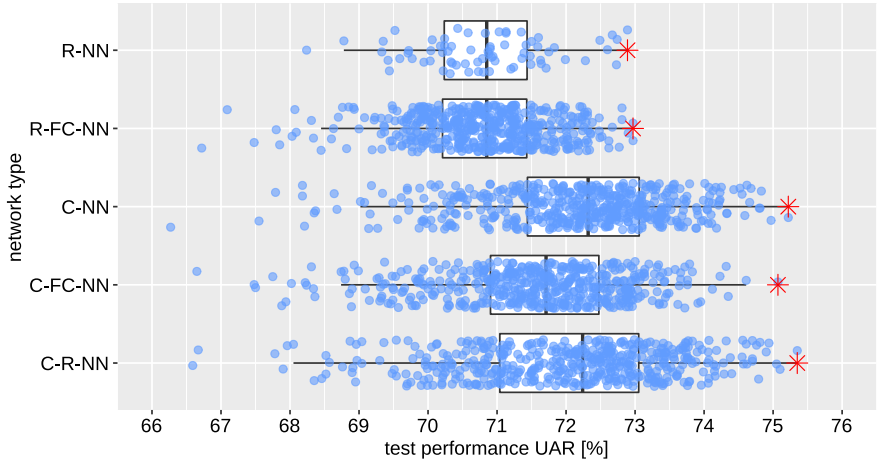

Figure 5.5: **Overall test performances of fine search sample.** Each data point indicates the test performance reached by a network configuration (averaged over three runs, respectively), grouped by network type. I applied a slight vertical jitter to facilitate the visibility of data points. Boxplots indicate the distributions of test performances. Red asterisk highlight the highest performance reached for each type.

5.3.3.2 Contest-like comparison

When designing the comparison between network types as a competition between approaches, each type is limited to contributing n configurations to be compared in terms of test performance. To select the most auspicious n configurations, each type's configurations were ranked by validation performance and the first n ranks were selected. Among the contributed configurations, only the **top configuration** prevails for each type, i.e. the one with the highest test performance [69, 150]. Fig. 5.6 shows the outcome of this competition for $n \in \{1, \ldots, 500\}$. Validation performance was measured by validation loss.

Fig. 5.6 shows that the maximum test performance increased with the number of contributed network configurations. This results from the test and validation performance not being perfectly correlated, which to some extent is inherent to splitting test and validation set. All ComParE challenge baseline systems [150] show a similar decorrelation effect. The performance difference between $n = 1$ and $n = 500$ was $< 1.5\,\%$ for all network types. The performance difference between the top configurations of convolutional

and recurrent networks (i.e. networks with and without convolutional stages) consistently stayed at $\approx 2\,\%$ regardless of the number of contributions. The lead between the top configurations of the network types R-NNs/R-FC-NNs and C-NNs/C-FC-NNs/C-R-NNs changed depending on n, respectively. For R-NNs and R-FC-NNs, the absolute performance difference between the top configurations stayed below 0.3 for any number of contributed configurations. For C-NNs, C-FC-NNs and C-R-NNs, the absolute performance difference between the top configurations started at $0.8\,\%$ for $n = 1$ and closed to $0.3\,\%$ for $n > 151$.

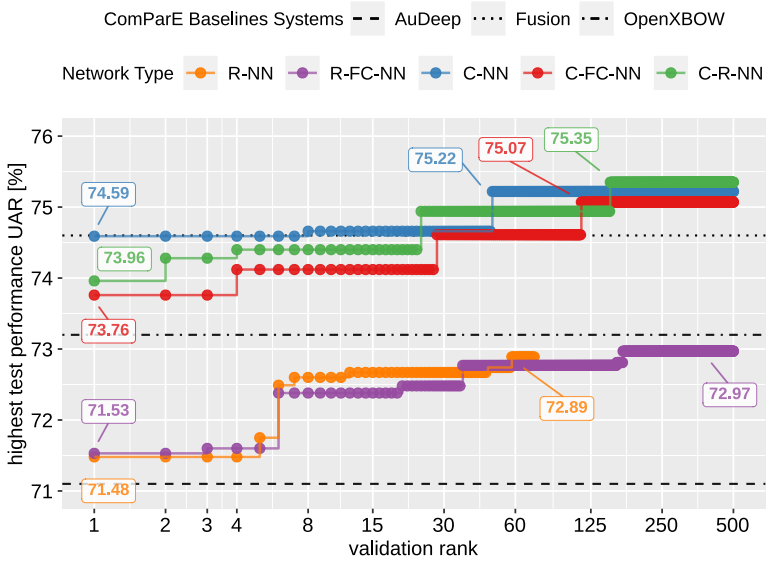

Figure 5.6: **Comparison between networks types in a contest-like setup.** First, all configurations were ranked by validation performance for each network type. The x-axis indicates the rank index n up to which configurations were contributed by each type. The y-axis indicates the highest test performance among all of each type's contributed configurations. Labels indicate performances at rank 1 as well as each type's overall highest performance. Horizontal dashed/dotted lines indicate baseline system performances.

Fig. 5.6 additionally indicates the performances of baseline systems from the ComParE 2018 challenge [150]. The AuDeep system with $71.1\,\%$ was selected for reaching the highest test performance among all previously published end-to-end systems. The OpenXBOW system with $73.6\,\%$ was selected for reaching the highest test performance among all previously published single systems, as well as conventional systems. The fusion system with $74.6\,\%$ was selected for reaching the highest previously published test performance overall. As shown, all network type's top configurations surpassed the AuDeep baseline and all network types with convolutional stages surpassed the openXBOW baseline. The Fusion baseline was surpassed by networks with convolutional stages depending on how many contributions one considers valid.

The parameter tuning for the baseline systems was performed by the challenge organizers [150]. They performed a grid search for one hyperparameter for each algorithm, respectively. In contrast to this study, the parameters were tuned directly on the test set, i.e. the validation performance was measured but not considered by the organizers. Choosing the algorithm settings with the highest respective validation rank reduces test UARs to 62.1 % for the AuDeep System and 67.6 % for the OpenXBOW system. The fusion system included the 3 algorithm configurations with the highest test performance. Consequently, the fusion system prematurely considered test performances at two steps: (1.) for selecting the included single-system algorithms, and (2.) for selecting the fusion method.

5.3.3.3 Significance testing of performance differences

I additionally tested the significance of the network type performance differences. The basic test methodology was to represent each model type through a group of its most m auspicious configurations and assess the test set performance between groups by means of a significance test.

To select the configurations into groups, I first ordered each model type's fine-search configurations by validation performance as in section 5.3.3.2. Among each type's first n validation ranks, I selected $m = 10$ models with the highest test performance. For example, for $n = 20$ the 10 configurations with the highest test performance among the first 20 validation ranks were selected. For significance testing, I chose the Wilcoxon signed-rank test for independent samples as it does not normal distributions in groups. Fig. 5.7 shows the resulting p-values for $n = \{10, \ldots, 500\}$ for each pairwise combination of network types.

For $n = 10$, all network types with convolutional layers (C-NN, C-R-NN and C-FC-NN) had significantly higher ($p < 0.01$) performance than networks without (R-NN and R-FC-NNs). The difference between C-NN vs. C-R-NN vs. C-FC-NN as well as R-NN vs. R-FC-NN was not significant. With growing n, the significance of the difference between networks with vs. without convolutional layers increased ($p < 0.001$). The difference between C-NN vs. C-R-NN and R-NN vs. R-FC-NN was not significant regardless of n. The difference between C-NN vs. C-FC-NN and C-R-NN vs. C-FC-NN varied between significant $p < 0.01$ and not significant $p > 0.01$ for $n > 20$.

To summarize, regardless of the amount of validation ranks considered, (1) convolutional networks reached a significantly higher performance than recurrent networks, (2) R-NNs and R-FC-NNs performed similar, and (3) C-NNs and C-R-NNs performed similar. The difference between C-NNs/C-R-NNs vs. C-FC-NNs was ambiguous, as the significance depended on the validation ranks considered.

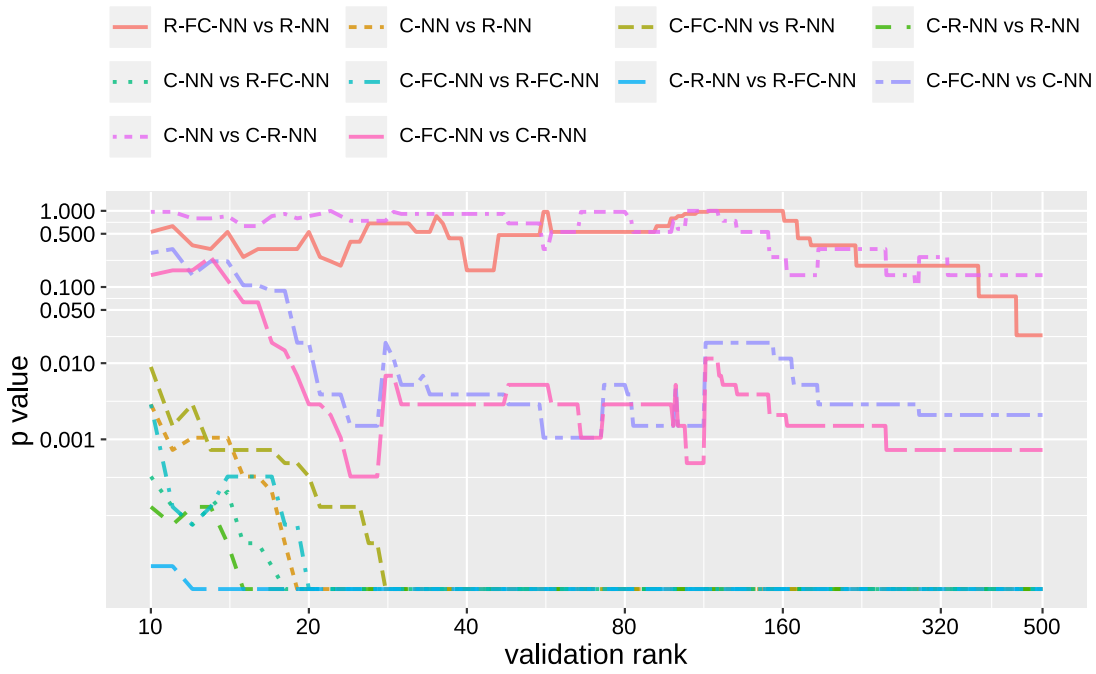

Figure 5.7: **Significances between network type performances.** The x-axis indicates the validation rank n for building network type groups. The y-axis indicates the p-value of the significance test between groups. Both x and y axis are log-scaled.

	Coarse Search	Fine Search
R-NN	temp. aggregation operation ∈ {GAP} ($R^2$0.34) → R-Stage rec. type ∈ {GRU} ($R^2$0.29) → R-Stage n. units first layer ≥ 24 ($R^2$0.18).	R-Stage bi-directtional ∈ {Yes} ($R^2$0.20).
R-FC-NN	FC-Stage batchnorm ∈ {No} ($R^2$0.4) → temp. aggregation operation ∈ {GAP} ($R^2$0.33) → R-Stage rec. type ∈ {GRU} ($R^2$0.17).	R-Stage num. blocks ≤ 3 ($R^2$0.045) → R-Stage n. units first layer ≥ 12 ($R^2$0.041) → R-Stage bidirectional ∈ {Yes} ($R^2$0.038) → FC-Stage activation ∈ {ReLU} ($R^2$0.075).
C-NN	temp. aggregation operation ∈ {GAP, GMP} ($R^2$0.39) → temp. aggregation operation ∈ {GAP} ($R^2$0.29) → freq. aggregation operation ∈ {GMP} ($R^2$0.23).	C-Stage num. blocks ≤ 4 ($R^2$0.13) → C-Stage batchnorm ∈ {Yes} ($R^2$0.07) → C-Stage pooling type ∈ {average, max} ($R^2$0.076) → C-Stage kernel shape = (1,5) ($R^2$0.1) → C-Stage activation ∈ {ReLU} ($R^2$0.26).
C-FC-NN	temp. aggregation operation ∈ {GAP} ($R^2$0.3) → FC-Stage batchnorm ∈ {No} ($R^2$0.38) → freq. aggregation operation ∈ {GAP, GMP} ($R^2$0.25).	freq. aggregation operation ∈ {GMP} ($R^2$0.04) → C-Stage num. blocks ≤ 3 ($R^2$0.06) → FC-Stage num. blocks < 2 ($R^2$0.15).
C-R-NN	temp. aggregation operation ∈ {GAP} ($R^2$0.16) → freq. aggregation operation ∈ {GAP, GMP} ($R^2$0.13) → C-Stage pooling type ∈ {average, max} ($R^2$0.14).	freq. aggregation operation ∈ {GMP} ($R^2$0.14) → C-Stage num. blocks ≤ 4 ($R^2$0.08) → R-Stage rec. type ∈ {GRU} ($R^2$0.09).

Table 5.3: **Best performing tree paths.** Each cell states the regression tree path leading to the leaf with the highest average performance for the respective network type. Splitting rules indicate stage name, parameter name and value according to Tab. 5.2. Arrows indicate the order of tree splitting rules. R^2 values indicates the gain in R^2 to the fit produced by a splitting rule, i.e. the split's effect size [171]. Abbreviations: *C-Stage*, *R-Stage* and *FC-Stage* abbreviate convolutional, recurrent and fully-connected stages.

5.4 Discussion

The main result of the network type comparison is that networks with convolutional stages (C-NNs, C-R-NNs and C-FC-NNs) outperformed recurrent networks (R-NNs and R-FC-NNs) by at least 2 % performance. This is supported by the outcome of the contest-like comparison as well as the significance testing. This result reflects recent developments in the DCASE community, in which virtually all competition systems contain convolutional stages [4, 36, 56, 66, 84, 108, 110, 153, 188]. However, the CP community still relies primarily on pure recurrent networks [67, 150, 177, 197].

Between C-NNs, C-FC-NNs and C-R-NNs there was no clear winner. In the contest-like comparison, the network type of the configuration with the best performance was dependent on the competition conditions, i.e. the number of contributions per type. Regarding the absolute evaluation metric, the final performance difference between the type's best overall configurations was just 0.3 %. The significance testing showed that C-NN and C-R-NNs performed similar regardless of the validation ranks considered. The difference between C-NNs/C-R-NNs and C-FC-NNs was ambiguous, as the difference was either significant or not significant when considering high or low validation ranks, respectively.

A limitation of this study was the infeasibility of evaluating the entire grid of network configurations. My results are based on randomly drawn samples of configurations and therefore should rather be interpreted as tendencies on the network type's potentials than as absolute upper limits.

Based on the results of the fine search, I hypothesize that among the convolutional networks, C-R-NNs have the highest potential for further improvement when continuing the parameter tuning process and C-FC-NNs the least. An indication for this is that C-R-NNs produced the highest density of top configurations in the range $> 74\,\%$ while also having the greatest remaining search space; C-FC-NNs produced the lowest density. However, this could also be an artifact of the random sample. Proof of this hypothesis needs further research.

The regression analysis of the parameter search showed that the most influential hyperparameters were the aggregation operations for reducing the tensor dimensionality inside the network. The best temporal aggregation operation was 1D GAP, regardless of the network type. This aggregation operation is already common in convolutional networks, as 2D GAP currently is the most prevalent layer for parallel frequency- and temporal aggregation [55, 60, 170]. However, recurrent networks commonly use the last step for temporal aggregation [134, 177, 197]. The best frequency aggregation operation was 1D GMP. These findings somewhat contradict current practices in network architecture design, since average pooling [55, 56, 60, 170] and flattening [22] are more commonly used for frequency aggregation.

Apart from the actual best aggregation operations found in our research, I draw the following conclusions: (1) selecting the right aggregation operation should be of primary concern in architecture optimization, and (2) for CNNs it should be considered to choose different aggregation operations for the frequency- and temporal dimension, as currently

most CNNs merely employ 2D global pooling operations [4, 55, 60, 170].

Based on the regression analysis of the random search results, I derive various recommendations for designing network architectures. In general, overly deep networks should be avoided as they reduce performance, which has already previously been documented [46, 55]. FCLs should not be used with batch normalization. For recurrent stages, I found bidirectional layers with GRU-Type units to work best. Pure recurrent networks also benefit from a larger number of units in the first layer, i.e. > 24. For convolutional stages, I recommend max or mean pooling for intermediate volume reduction rather than striding kernels, 1D Kernel sizes, and ReLU activation function. The best performing tree path of the fine search analysis for the C-NN type support this finding. For C-FC-NNs and C-R-NNs, similar rules are found among various regression tree paths, although not the best performing ones. Most of these findings reflect the configurations of conventional convolutional networks such as VGGNet. However, the superiority of 1D kernel sizes was noteworthy, as currently 2D quadratic kernel sizes are more common [4, 22, 56, 60].

A result that might be considered surprising is that the most performant network configurations were not the most "advanced" ones: For example, shallow C-NNs with 1D Kernel sizes and simple temporal average pooling were among the best performing networks. These are arguably simpler than other more complex architectures covered in the search space, e.g. C-R-NNs with ResNet-like conv. stages. I hypothesize that this is attributed to the small data set size, which is prone to overfitting, putting more complex layer types at a disadvantage as these usually contain more trainable parameters.

Finally, our study indicates that components are most important to differentiate vocalization classes. The results suggest that the information is primarily encoded in qualitative characteristics of local spectrogram patches rather than the temporal evolvement of the signal. This is supported by the finding that R-NNs, which primarily process sequential temporal information, were outperformed by C-NNs with 1D kernel sizes, which calculate local features on STFT-like time frames without incorporating temporal information and forward a mere statistical summary of these features to the output layer. Consequently such convolutional networks fundamentally work similar to conventional classification approaches such as Open SMILE [38, 150], which likewise perform classification on statistics (*functionals*) of LLDs (features computed on STFT frames).

CHAPTER 6

Study (C): Detailed Investigation of CNNs for Automatic Classification of Infant Vocalizations

6.1 Study goal

The goal of this study was to investigate the influence of the architecture configuration of computer vision-like CNNs[1] for infant vocalization classification in detail. More precisely, the goal was to identify the networks properties with the greatest association to the performance.

The similarities between both studies are: (1) both investigate infant vocalization classification, (2) both employ similar processing pipelines, e.g. using log Mel-scaled spectrograms, (3) both core study objectives focus the network architecture. However, while study (B) presented a broader comparison of various network types, this study specifically focused on CNNs in depth. CNNs were identified in study (B) as the network type with the highest performance potential. Additionally, the classification challenge itself was broader, as the label set included five vocalizations classes instead of three.

As study (B), this study is of importance to further the understanding of representation learning systems for automatic infant vocalization classification. As CNNs with computer vision-like architectures have become particularly prevalent in ASR (see state of research section 3.2.3), studying the configuration of such architectures is beneficial for ASR in general.

[1] Contrary to study (B), the term CNNs refers to the usual definition of CNNs as neural networks with convolutional layers in general (see foundations section 2.3.4).

The methodology was as follows: I first defined a CNN architecture scheme representative of conventional VGGNet-like CNNs [155]. I produced numerous configurations of this scheme and performed a grid search. I analyzed the association between architectural CNN properties and the classification performance through statistical methods to identify the most influential properties.

Besides the intersections with study (B), this study incorporates results form study (A), as the target classes were derived from one of the resulting classification scheme.

6.2 Materials and methods

The structure of the methods section is as follows: Section 6.2.1 presents the acoustic data set. Section 6.2.2 presents the general classification pipeline involving the CNN. Section 6.2.3 presents the CNN architecture scheme and configuration search space. Section 6.2.4 and 6.2.5 present the training and evaluation setup, respectively. Finally, section 6.2.6 presents the experimental setup, i.e. the approach for investigating correlations between network properties and performance.

6.2.1 Acoustic data set

The data set of this study originated from study (A) (see study (A) section 4.2.2). The source signals were the same 228 sound files, collected from free online sources, cut into 883 segments. These segments are referred to as *examples* in this study.

I labeled all sound files according to the classification scheme shown in Tab. 6.1. Each example was assigned to exactly one class. This classification scheme was based on the results of study (A) (see study (A) section 4.3.5). However, I slightly modified the scheme by grouping neighboring classes. I grouped *crying* and *whining* (from study (A)'s scheme) into *fussing* (in this scheme), and *babbling* and *squealing* into *babbling*. This change ensured that all classes had sufficient examples.

The classes *fussing* and *crying* are frequently required for distress and pain assessment in medical applications [70, 104, chapter 37]. The classes *babbling*, *laughing* and *vegetative vocalizations* are typically of interest in assessment of vocal development [21, 123] (see state of research section 3.3.1).

Key property of the data set are: (1) it considers *general* infant vocalizations, instead of just a subdomain, such as cry-like vocalizations, (2) examples involve a greater temporal context than most studies on automatic vocalization recognition. I consider examples to contain *sequence* level information, rather than *unit level* information, as discussed in section 4.2.2. [2, 23, 43, 87, 182, 196].

Section 2.4.1 discussed how classification algorithms can be adapted for detection tasks, which in principle applies to this study as well. However, the classification scheme does not contain a class for complete absence of infant vocalizations. Experiments showed that examples with near silence are mapped to the *vegetative vocalizations* class, due to the examples with slight breathing being most similar acoustically.

Class name	Perc. affective valence	Qualitative description
Crying	highly negative	Vocalizations of high negative valence and distress. Continuous, periodic and intense.
Fussing	moderately negative	Vocalizations of moderate negative valence. Similar to crying, but quieter and more discontinuous.
Babbling	neutral or positive	Vocalizations of neutral or positive valence. These fall under the *protophone* umbrella term in vocal development research [21], e.g. *protophones*, e.g. canonical babbling, squealing, grunting, moaning, gooing, etc.
Vegetative vocalizations	neutral	Involuntary vocalizations of neutral valence, e.g. coughing, sucking, burping etc. Also, near silence with slight breathing.
Laughing	positive	Full-lunged laughing as well as slight chuckling.

Table 6.1: **Target classes.**

6.2.2 Classification pipeline surrounding the neural network

The task was modeled as a monophone sound classification task with five classes. As in study (B), targets were one-hot encoded for training and integer-encoded for validation (see foundations section 2.4.1 on these terms). The classification pipeline generally corresponds to the on in study (B) (see study (B) section 5.2.2), which in turn builds on the general framework presented in foundations section 2.4.3:

1. **Preprocessing:** The preprocessing directly corresponds to the procedure of study (B). The sampling rate was 22 050 Hz.
2. **Feature map conversion:** The calculation procedure was equal to the one in study (B). However, here I used the power spectrogram instead of the magnitude spectrogram (see foundations section 2.1.1). The exact parameters of the feature map were as follows: The STFT length was $N = 2048$ samples, window length of 40 ms, hop length of 20 ms, a hanning-window function, and the number of Mel filters was 80. Spectrograms were z-standardized with global statistics calculated on the training set. The feature extraction was implemented with the python library `librosa v.0.7` [103].
3. **Segmentation:** In this study, the segment size and hop size was part of the investigated hyperparameters, see section 6.2.3.
4. **Classification:** Directly corresponds to study (B).
5. **Output aggregation and binarization:** Directly corresponds to study (B).

6.2.3 Neural network architecture scheme and search space

Figure 6.1 shows the CNN architecture scheme for prediction. Table 6.2 shows the corresponding scheme hyperparameters. As in study (B), the actual architecture of a network results indirectly through the architecture build rules defined in the scheme and hyperparameters. Hyperparameters are *not* chosen directly for layers (see study (B) section 5.2.3). For further explanations on the layers and concepts in this section, see foundations section 2.3 and particularly section 2.3.2.

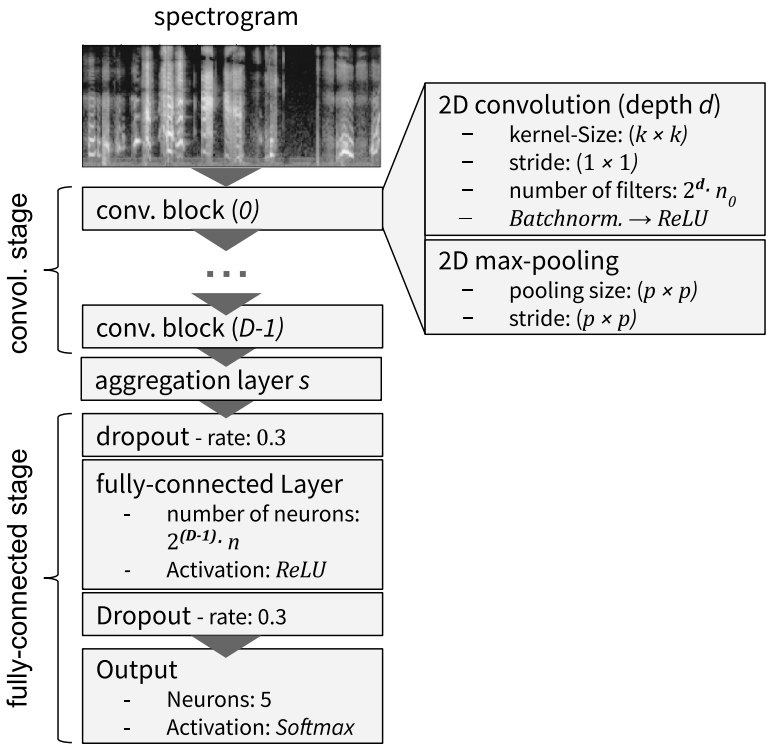

Figure 6.1: **CNN architecture scheme.** The values of the parameters D, k, p, n_0, τ are subject to the configuration. Table 6.2 explains the parameters. The variable d indicates the current depth of the convolutional block $d \in \{0, \ldots, D-1\}$

The scheme was designed to represent a template for "conventional" CNN architectures used for computer vision, such as LeNet, AlexNet, or VGGNet (see foundations section 2.3.4 for these architectures). It was aimed to keep those properties of the network to be variable that I hypothesized to be of primary interest for increasing performance and those fixed that were of less interest. As these networks, the scheme contains a convolutional and a fully-connected stage: The convolutional stage extracts acoustic features from the spectrogram via alternating convolutional and pooling layers, and the fully connected stage processes those features to the output. The aggregation layer adapts the volume dimensionality between both stages, by aggregating the time and frequency axis. The quadratic kernel shapes reflect that the architecture treats the

temporal and frequency axis identically as though they were spatial dimensions of an image.

All scheme hyperparameters force layers to share certain properties, such as the kernel size. The number of units doubles with each layer as in VGGNet [155]. Therefore, the number of units in a layer is determined indirectly through the parameters *filters in the first layer* n_o and *depth* D.

Justifications for the scheme hyperparameters and ranges are as follows:

- The depth, kernel size, and filters in the first layer primarily determine the capacity of a CNN according to Goodfellow et al. [46, chapter 11].
- The aggregation layer proved to be one of the most important hyperparameters in study (B). It is one of the central architectural differences between conventional CNNs, such as VGGNet, and more modern CNNs, such as ResNet [55, 155].
- Although pooling sizes greater than (2×2) are uncommon in image recognition networks, larger pooling sizes are common in networks for audio recognition, e.g. the DCASE baseline systems [110].
- The segment length determines the temporal context evaluated by the network at once. The values correspond to approximately the average duration / half of the average duration of the database examples (see Tab. 6.3).

The segmentation hop size in the processing pipeline was adapted dynamically according to the chosen segment length T_{seg}. For $T_{\text{seg}} = 3\,\text{s}$ I distributed 2 segments and for $T_{\text{seg}} = 6\,\text{s}$ I distributed 3 segments across each example's entire duration. These numbers of segments were chosen as the minimum possible values to cover the longest examples in the database entirely with the respective context length.

This scheme might be considered a special case of the global scheme presented in study (B) (see study (B) section 5.2.3). In the terminology of study (B), this was a C-FC-NN. The central differences between this scheme and study (B)'s scheme were: (1) the recurrent stage was omitted, (2) the frequency and temporal aggregation layers were fused to a single aggregation layer, and (3) some hyperparameters were fixed that were variable in study (B), including: the activation functions were fixed to ReLU, the unit growth factor was fixed to 2, there was one fixed FCL, max pooling was the fixed intermediate pooling operation, and convolutional layers were always followed with batch normalization. All of these fixes are in line with the design principles for conventional CNNs, such as LeNet, AlexNet, and VGGNet.

Consequently, this scheme is more restricted than the global parent scheme of study (B), i.e. there are less degrees of freedom to be investigated in a hyperparameter search. The reason for this was to keep the search space small enough to perform an exhaustive grid search (see foundations section 2.2.2.3), focused on the central parameters of this CNN type.

Parameter name	Parameter values	Description
aggregation layer s	$s \in \{\text{flatten}, \text{GAP}\}$	Determines the layer that adapts the volume dimensionality of the convolutional stage to the fully-connected stage. *Flatten* flattens the volume to a vector. GAP averages globally across the time and frequency axis. When using GAP, the last pooling operation is omitted to avoid the consecutive application of two pooling layers.
segment length T_{seg}	$T_{\text{seg}} \in \{3\,\text{s}, 6\,\text{s}\}$	temporal length of the segments
depth D	$D \in \{2, 3, 4\}$	number of convolutional blocks
kernel size $(k \times k)$	$k \in \{3, 5, 7\}$	Determines the kernel size common to all convolutional layers. The stride is always (1×1).
pooling size $(p \times p)$	$p \in \{2, 3, 5\}$	kernel size and stride of 2D max pooling layers
filters in the first layer n_0	$n_0 \in \{16, 32, 64\}$	amount of filters in the first convolutional layer

Table 6.2: **Network scheme hyperparameters and search spaces for configuration of the architecture scheme shown in Fig. 6.1.**

6.2.4 Training setup

The training setup corresponded to study (B) (see study (B) section 5.2.5). The learning rate was 0.0001, the batch size 64, and early stopping patience was 30 epochs. For implementation of networks and training I used the python library `keras v.2.2.4` with `tensorflow-gpu v.1.14.0` backend.

6.2.5 Evaluation setup and performance metric

For evaluation I chose a nested cross-validation setup to compensate sampling bias due to the small data set size (see foundations section 2.2.4.6). Contrary to study (B) and (D), I was free to chose the evaluation setup as the data set was created specifically for this study.

I split the data set into 4 folds. Assignment from examples to folds was performed in a stratified manner, i.e. each fold was aimed to contain similar distributions of class examples. Folds always contained all signals from the same source signal to avoid the *album effect* (see foundations section 2.4.4). Table 6.3 summarizes the example distributions for all folds.

In each evaluation loop, two folds were used in combination for training, one for validation, and one for testing. The validation set was solely used for determining the stopping epoch through early stopping. However, validation performance itself was of no interest in this study. Consequently there were $4 \cdot 3 = 12$ evaluation runs per CNN configuration, so that each fold was used thrice for testing, and each of the remaining folds once for validation. The total performance of a network configuration was the average of all 12 runs.

Fold:	1	2	3	4	$\sum$	durations
babbling	55	63	59	79	256	6.24 ± 0.79 s
crying	65	64	78	74	281	6.50 ± 0.88 s
fussing	38	38	43	38	157	6.26 ± 0.84 s
laughing	10	12	11	8	41	5.85 ± 0.90 s
vegetative	25	30	19	24	98	5.76 ± 0.89 s
$\sum$	193	207	210	223	833	6.26 ± 0.87 s

Table 6.3: **Overview over data set folds.** Table cells state the number of examples for each class and fold. The last column indicates the mean and standard deviation of the example durations, measured in seconds.

The performance metric was UAR (see foundations section 2.2.4.3) as in study (B), as it accounts for the imbalance in class distribution and is the standard metric in CP competitions [148].

6.2.6 Experimental setup

The experimental setup fundamentally corresponded to a grid search (see foundations section 2.2.2.3), i.e. all possible architectures resulting from the search space defined in Tab. 6.2 were trained and evaluated. I omitted overly large network configurations in light of the low amount of training data, i.e. omitting configurations with more than 5 million trainable parameters. The total amount of network configurations was 283.

The goal was to identify correlations between network properties and performance. As in study (B), those network properties encompassed the scheme hyperparameters themselves as defined in Tab. 6.2. The results section refers to these parameters as the **determining CNN properties**.

However, in this study I additionally included derived network properties that capture interaction effects between the scheme hyperparameters. This was done to gain greater insight into the relationship between network architecture and performance for CNNs specifically. The results section refers to these parameters as the **derived CNN properties**. They were as follows:

- **Trainable Weights:** Total amount of trainable weights in the CNN. This is considered as one of the central properties for a network's capacity [46, chapter 11].
- **Input size of the fully-connected layer (FCL)**: Size of the vector input to the FCL after the aggregation layer, i.e. the size of the *feature bottleneck*. This number indicates the amount of acoustic features that is finally extracted by the convolutional stage. When the aggregation layer is GAP, the size is $2^{D-1} \cdot n_0$ (variable names correspond to Tab. 6.2). When the aggregation layer is *flatten*, the size is

$$\frac{m \cdot T_{\text{frames}} \cdot 2^{(D-1)} \cdot n_0}{p^{2D}} \ ,$$

where $m = 80$ is the number of input Mel bands and T_{frames} is the number of input spectrogram time frames. This feature captures the interaction between network depth and width (see foundations section 2.3.1 for these terms).

- **Cumulative receptive field:** This property indicates the range each filter of the last convolutional layer (before the FCL) receives across the time and frequency axis through all of its preceeding layers. The range is indicated in units [number of Mel bands] for the frequency axis and in [seconds] for the time axis. I calculated this value as proposed by Le and Borji [89]:

$$\text{RAN}_u = \text{RAN}_{u-1} + (q_u - 1) \prod_{i=0}^{u-1} s_i \ ,$$

where RAN_u is the range of layer $u \in \{0, \ldots, 2 \cdot D - 1\}$ (including convolutional as well as pooling layers), q_u the the layer kernel size, and s_i the layer stride. I measured the cumulative receptive field for the time and frequency axis separately. Koutini et al. [83] showed the cumulative receptive field to be a factor for performance in the context of acoustic scene classification.

Contrary to study (B) and (D), this study contained a single experimental round, i.e. there was no secondary step that build on intermediate results. I chose the following methods to analyze the correlations between network properties and performance: (a) PCA, which is a dimensionality reduction technique for correlated variables, which I applied to infer correlations through factor loadings, as well as to visualize them [69, chapter 6, 10], and (b) direct measurement of correlations through correlation metrics, globally as well as in selected subgroups. For clarity, the results section explains these methods in greater detail.

6.3 Results

The structure of the results section is as follows: Section 6.3.1 presents the overall system performance and the analysis the class-wise performance. Section 6.3.2 presents the more detailed analysis on the relationship between network properties and performance.

6.3.1 Overall system performance

Figure 6.2 summarizes the distribution of test performances of all 283 tested CNN architectures through a histogram. The mean performance was 64.83 % with a standard deviation of 4.29.

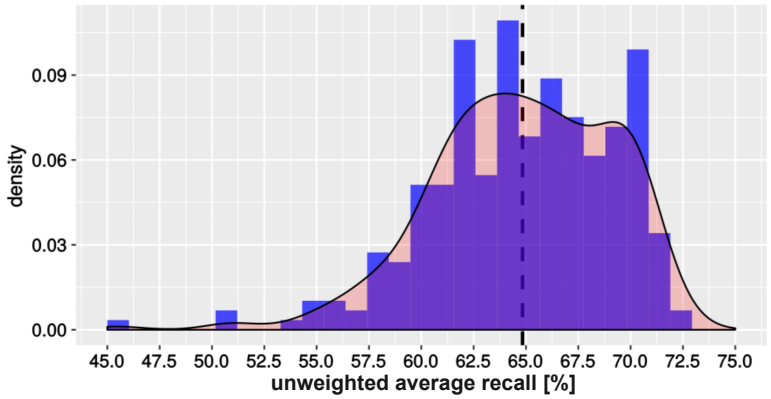

Figure 6.2: **Histogram of system performances**. The x-axis indicates the system performances as measured with the UAR. The y-axis indicates relative amounts of CNN configurations. The blue bars indicate histogram counts. The red area indicates the smoothed density estimate. The vertical black line indicates the mean performance of all CNN configurations.

The highest performance value was 72.84 %. The corresponding configuration was: aggregation layer s = GAP, temporal context $T_{\text{seg}} = 6\,\text{s}$, depth $D = 3$, kernel size $k = 3$, pooling size $p = 2$, number of filters in the first layer $n_0 = 32$.

Fig. 6.3 shows the corresponding confusion matrix for the entire test set, as produced by this best performing CNN configuration. To ensure that each test fold was included exactly once into the confusion matrix (as each one was used thrice for testing, see section 6.2.5), for each test fold I chose the result that yielded the lowest validation loss.

The matrix shows that the most often confused classes were fussing and crying, fussing and babbling, babbling and vegetative vocalizations, and laughing and babbling.

6.3.2 Relation between CNN properties and performance

This section presents a detailed analysis of the association between CNN properties and the performance. Section 6.3.2.1 visualizes property importance by means of a principal component analysis. Section 6.3.2.2 analyzes significances and effect sizes of the CNN properties.

6.3.2.1 Principal component analysis

Principal component analysis (PCA) is a method for dimensionality reduction that approximates a larger set of correlated variables through a smaller set of decorrelated variables representing linear combinations of the former [69, chapter 6, 10]. I included all CNN properties described in section 6.2.6, i.e. the determining CNN properties (the scheme hyperparameters 6.2), the derived CNN properties, and the performance metric UAR.

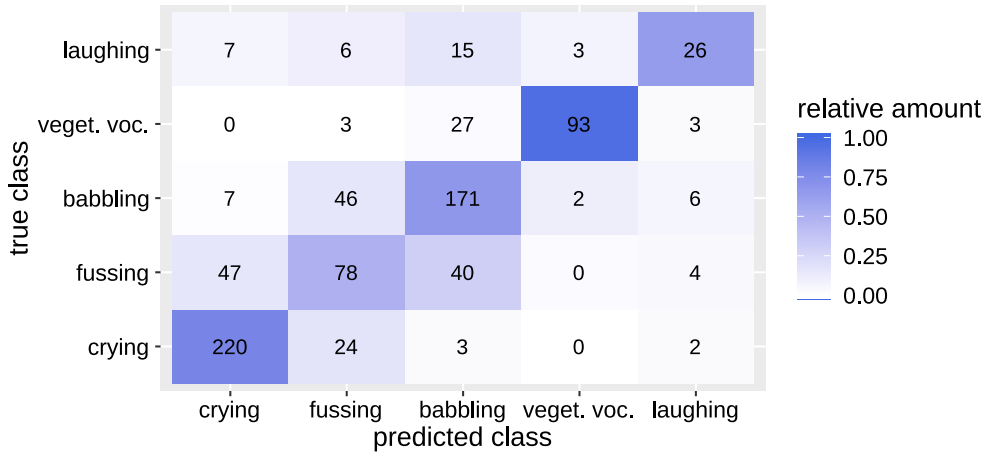

Figure 6.3: **Confusion matrix for all audio examples.** The numbers show absolute classification results. Colors indicate classification results normalized per row, i.e. the amount of predicted classes in relation to the amount of examples present in the respective true class.

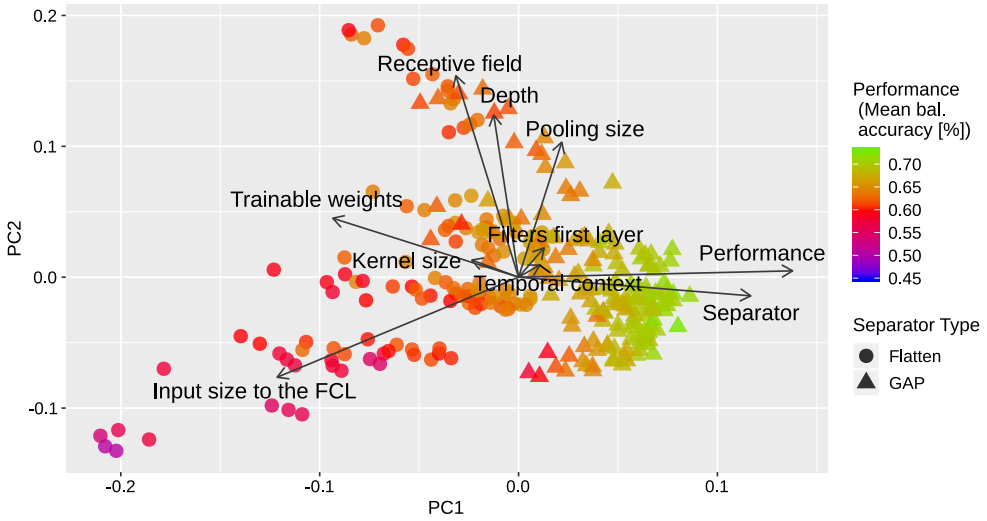

Figure 6.4: **PCA visualization**. The axis show the first two principal components. Data points represent CNN configurations, which are placed according to their PCA rotations. Data points colors and shapes code the performance and aggregation layer, respectively.

The only correlations exploitable by the PCA were between the three groups of variables (a) determining CNN properties (b) derived CNN properties, and (c) the performance; However, within the group of determining CNN properties there were no exploitable correlations, as these were defined to be independent of one another. Consequently, the PCA primarily reveals any correlations between the performance and network properties of any kind.

The binary categorical variable for the aggregation layer was dummy-encoded with $0 \mathrel{\hat{=}}$ *flatten* and $1 \mathrel{\hat{=}}$ GAP. All variables were z-standardized. The PCA was performed with the `prcomp()` function of the `R` package `stats v.3.4.4`.

Table 6.4 lists the variable loadings of the first two principal components and figure 6.4 visualizes the projections of the CNN configurations. The relatively low cumulative explained variance of 49 % is due to the variables of the determining parameters group being decorellated.

The first principal component assigns the highest variable loadings to the performance, the input size of the FCL and the aggregation layer. Consequently this component represents properties important to the classification performance, with GAP aggregations as well as smaller sizes to the FCLs positively impacting performance. A decreased number of trainable weights also increased performance, albeit to a lesser degree.

The second component was associated with CNN properties correlated with the cumulative receptive field, which were the pooling size and depth. The small loading of the performance variable indicates that these properties have little impact on the CNN performance.

Variable name	Variable loading	
	PC1	PC2
aggregation layer	**0.483**	−0.059
segment length	0.044	0.038
depth	−0.052	**0.512**
kernel size	−0.097	0.054
pooling size	0.09	**0.426**
filters first layer	0.053	0.092
trainable weights	−0.386	0.186
input size of the FCL	**−0.502**	−0.316
cum. receptive field	−0.13	**0.636**
performance	**0.57**	0.020
variance explained	24.41 %	19.11 %

Table 6.4: **PCA variable loadings** for the first two components. Variables with loadings > 0.4 are printed bold.

6.3.2.2 Analysis of network property significances and effect sizes

As PCA variable loadings are not interpretable in absolute terms, I additionally calculated significances and effect sizes for CNN properties.

The correlation was calculated univariate for each property and the performance separately. For all continuous network properties, I chose the Spearman correlation coefficient for measuring the effect size r and the significance p. For the categorical feature *aggregation layer* I chose the point-biserial correlation coefficient instead, with dummy-encoding $0 \mathrel{\hat{=}}$ flatten and $1 \mathrel{\hat{=}}$ GAP. According to Sullivan and Feinn [168], properties with significance values of $p < 0.01$ and effect sizes $|r| > 0.2$, > 0.5 and > 0.8 might be interpreted has having *small*, *medium* and *high relevance*, respectively.

Spearman correlation values were calculated via the function `cor()` and significance values via the function `cor.test()` of the `R` package `stats v.3.4.4`. The point biserial correlation was calculated via the function `biserial.cor()` of the `R` package `ltm v.1.1-1`.

Main effects

The left column of table 6.5 shows effect sizes and significances when taking all CNN configurations into consideration. The properties with the highest effect sizes were the aggregation layer and the input size of the FCL, both having medium relevance. This reflects the variable loadings of the first principal component presented in section 6.3.2.1. Increasing the number of filters in the first layer and dercreasing the number of trainable weights had small relevance. All other CNN properties had no relevance.

Effects grouped by aggregation layer

I additionally calculated significances and effect sizes for CNN configurations grouped by aggregation layer. This was motivated by the observation that the aggregation layer and the input size of the FCL had the largest main effects while also being intercorrelated, so I aimed to investigate if this effect still held up in subgroups of aggregation layers. The two right columns of table 6.5 show effect sizes and significances for those subgroups.

For the *flatten* aggregation subgroup, the properties with the highest effect sizes were the pooling size, the input size of the FCL and the cumulative receptive field, all having medium relevance. Larger pooling sizes, smaller input sizes to the FCL and larger cumulative receptive fields positively affected the performance. Those properties were also intercorrelated, i.e. larger pooling sizes mainly produce smaller input sizes to the FCL and larger cumulative receptive fields. All other network properties had small relevance, except the kernel size.

For the GAP aggregation subgroup, all properties had either no or small relevance. Consequently, these CNNs were overall less sensitive to parameter variations than CNNs with *flatten* aggregations. Among those properties with small relevance, the most significant was the cumulative receptive field, with smaller cumulative receptive fields positively affecting the variance. This was contrary to CNNs with *flatten* aggregations, which benefited form larger cumulative receptive fields.

Grouping variable	Main effects	aggregation	
Grouping value	-	Flatten	GAP
aggregation layer	*** **0.631**	–	–
temporal context	0.038	**−0.255	** 0.232
depth	−0.030	** 0.276	**−0.232
kernel size	−0.114	−0.173	*−0.182
pooling size	0.018	*** **0.552**	−0.103
filters first layer	*** 0.284	*** 0.377	** 0.228
trainable weights	***−0.381	**−0.257	*−0.159
input size of the FCL	***−**0.604**	***−**0.630**	−0.029
cum. receptive field	*−0.119	*** **0.518**	***−0.298

Table 6.5: **Effects of network properties.** Table cells indicate effect sizes as measured with the spearman / point-biserial correlation coefficient. Asterisks indicate p-values with *** $= p < 0.001$, ** $= p < 0.01$, * $= p < 0.05$. Effect sizes with at least medium relevance > 0.5 are highlighted bold. The left column shows the main effects. The two columns show effects when grouping CNNs by aggregation layer.

Fig. 6.5 visualizes the association between CNN properties and the performance for aggregation subgroups. Only properties with at least small relevance in either subgroup are shown. I highlight the following observations:

1. **Aggregation layer:** GAP aggregations clearly outperform *flatten* aggregations on average.
2. **Input size of the FCL**: There is a visible overall negative association between the input size of the FCL and the performance. GAP aggregations are shown to cause smaller input sizes to the FCL in general. The local maximum for *flatten* aggregations is at ≈ 256 – 2048 and for GAP aggregations it is at ≈ 64 – 128.
3. **Depth:** The optimal depth for both subgroups is 3.
4. **Cumulative receptive field:** Both aggregation layers cause opposite correlations between the size of the cumulative receptive field and the performance. The local maximum for GAP aggregation layers is at ≈ 30 Mel frequency bands and ≈ 0.65 s. For *flatten* aggregations, it is at ≈ 150 – 400 frequency bands and ≈ 3 – 6 s. For the frequency axis, this corresponds to ≈ 2 – 5 times of the provided input range of 80 Mel frequency bands. For the time axis, this corresponds to the entire temporal context provided by the input segments.

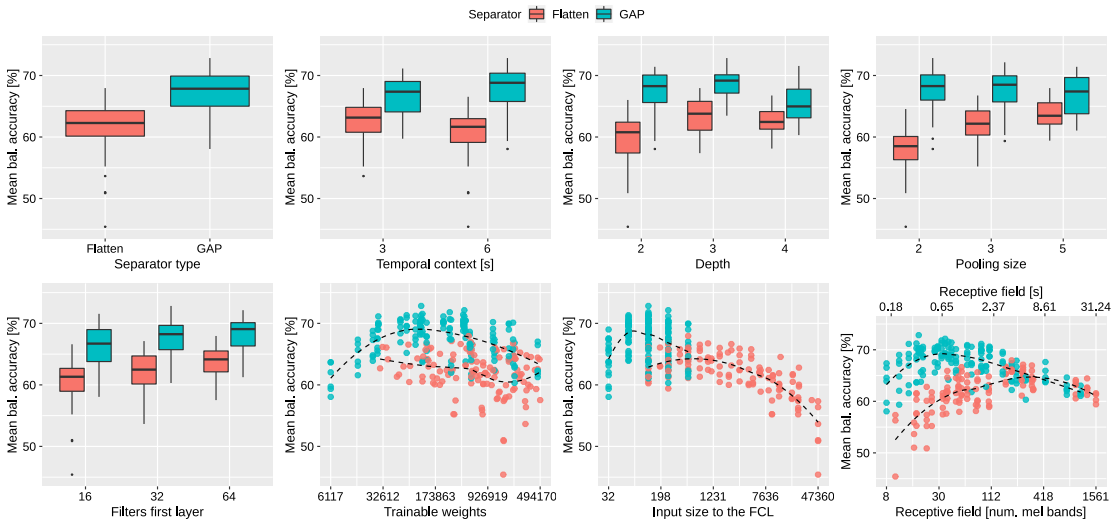

Figure 6.5: **Plots of CNN properties vs performance.** CNNs are color coded by aggregation layer. The black dotted lines of the last three plots show local polynomial regression lines fitted to the subgroups.

6.4 Discussion

I summarize that the CNN properties with the strongest overall impact on the performance were the choice of aggregation layer as well as the input size of the fully connected layers. The most efficient CNN configurations employed a 2D GAP aggregation layer and had cumulative receptive fields of $\approx 0.65\,\mathrm{s}$ and $\approx 37\,\%$ of the provided frequency range. CNNs with *flatten* aggregation layers additionally benefited from large pooling kernel sizes/strides in conjunction with large cumulative receptive fields.

I conclude that CNNs optimized for the task of infant vocalization classification must be designed to have small *feature bottlenecks* between the convolutional stage and the fully connected stage. This bottleneck should be achieved through broad aggregation of the convolutional feature maps of the convolutional stage across the time- and frequency axis. This aggregation should be performed "late", i.e. once at the end of the convolutional stage through a broad global pooling layer such as GAP, in contrast to aggregation through repeated broader pooling between convolutional layers. The second most important factor is management of the size of the cumulative receptive field, appropriate to the aggregation layer. Koutini et al. [83] made a similar observation for acoustic scene classification.

Increasing the depth of the CNN was less effective, although it also produced a smaller feature bottleneck. However, the architecture scheme doubled the amount of filters when increasing depth, leading to diminishing returns when increasing depth to produce a smaller bottleneck. Consequently it could be beneficial keep the amount of filters constant across all layers, despite this being an uncommon CNN architecture concept. Increasing the amount of filters in the convolutional layers affected the performance

positively despite producing larger feature bottlenecks. Possibly detrimental effects of larger amounts of filters were not fully explored as I excluded CNNs with more than 5 million trainable weights, excluding the largest CNNs.

The implication of the dominance of GAP aggregation layers is that this layer provides a sort of regularization that improves the generalization ability of the networks without sacrificing audio properties important for class differentiation. Particularly information about the temporal progression of filter outputs is withheld from the fully connected layers, which are instead provided with a statistical summary about local acoustic properties calculated at various time steps. Consequently, vocalization classes are differentiable through statistical summaries of local properties alone. This reflects the approach of conventional ML approaches for infant vocalization recognition, which also evaluate statistical summaries of local acoustic properties, such as the conventional baseline systems of ComParE challenge [150].

I highlight that networks with *flatten* aggregations had the highest performance when each filter of the last convolutional block covered a range of $3-6$ s. This means that the CNN filter cumulative receptive field effectively covered the entire input spectrogram. The temporal progression of the input signal was thus captured by the filter stage, instead of being evaluated by the fully connected stage. This hints at the fully connected layers being inadequate for interpreting local acoustic properties.

Finally, there are interesting similarities between the confusion matrix and the classification scheme the target classes were based on. The classification scheme originated from the results of study (A) (see study (A) section 4.3.5). I highlight the following observations: (1) Class similarity according to human perception is reflected in the misclassification by the algorithm. Evidence for this is that classes were more often confused if they were neighboring according to the classification scheme. This neighborhood association is primarily determined by the valence rating. (2) The class with the lowest accuracy was *fussing*. This is class acts as a valence "bridge" between babbling and crying, but has few unique acoustic properties besides that (as opposed to laughing for example). This underlines the hypothesis of *crying as a graded signal*, i.e. that distress vocalizations form a continuum, which is difficult to quantize into discrete classes [16].

CHAPTER 7

Study (D): Compensating Class Imbalance for Acoustic Chimpanzee Detection With Convolutional Recurrent Neural Networks

7.1 Study goal

The goal of this study was to investigate methods for compensating severe class imbalance for automatic detection of chimpanzee calls in long-term audio recordings. The task was to detect chimpanzee calls *drumming* and *vocalizations* in PAM recordings from the African rain forest (Taï National Park, Côte d'Ivoire). The data set contained a severe imbalance between positive and negative classes: The test set, with a total duration of 179 hours, contained merely $\approx$ 10 minutes of chimpanzee calls. This imbalance is representative of the prevalence of chimpanzee calls in naturalistic recordings. Consequently, I selected a set of common methods for compensating this imbalance, which were: (1) spectrogram denoising, (2) resampling, and (3) alternate loss functions. The core question was which of these methods is most effective for increasing performance. An additional question was whether frame-precise detection through CRNNs is feasible. Both of these questions refer to the research gaps discussed in section 3.3.3.

The study is of importance for transferring ML detection systems to realistic conditions: Severe rarity of target class is known to negatively impact the performance of ML systems, as these tend to be biased towards the majority class Johnson and Khoshgoftaar [72]. Previous studies circumvented the imbalance issue through manual rebalancing of their data sets. This relates to the research gaps discussed in section 3.3.3.

The detection approach was adapted by Cakır et al. [22]. Their approach is based on a CRNN with frame-wise outputs. I extended their detection pipeline through various stages aimed at compensating class imbalance. I studied the effects of these stages on the detection performance in two experimental rounds. The main tools for analysis were regression trees and statistical assessment, similar to studies (B) and (C).

7.2 Materials and methods

The structure of the methods section is as follows: Section 7.2.1 presents the acoustic data set. Section 7.2.2 presents the general detection pipeline involving the CRNN. Section 7.2.3 presents the approach to spectrogram denoising. Section 7.2.4 presents the CRNN architecture. Section 7.2.5 presents the loss variants and section 7.2.6 the resampling variants. Section 7.2.7 and 7.2.8 present the training and evaluation setup, respectively. Finally, section 7.2.9 presents the experimental setup.

7.2.1 Acoustic data set

The data set was originally collected by Kalan et al. [74, 75] with the aim of developing an automated approach for detecting primate calls in PAM forest recordings [57]. Heinicke et al. [57] presented the evaluation of this automated system.

The recording site was the western section of the Taï National Park, Côte d'Ivoire. The area sampled the territories of two chimpanzee communities. The soundscape of the park featured a wide variety of biogenic sounds, e.g. birds, insects, anthropogenic sounds, rain, wind etc. The recording setup comprised 20 ARUs distributed evenly across an area of $\approx 35\,\mathrm{km}^2$. ARUs recorded in stereo with a sampling rate of 16 kHz and a depth of 16 bit. The recording took place from from November 2011 to May 2012. ARUs recorded daily from 6 am to 6 pm at the full hour for 30 minutes. Overall 12 889 h of audio data was collected.

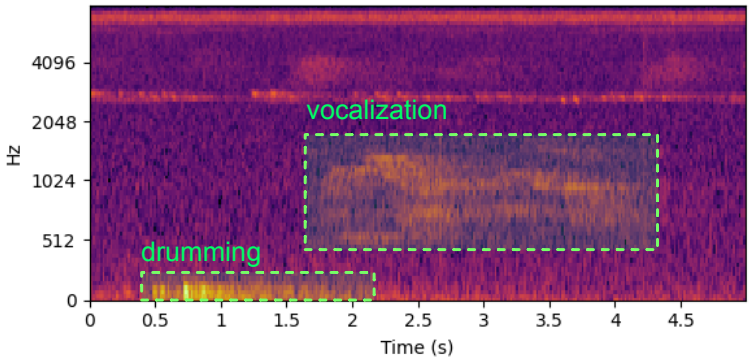

Figure 7.1: **Example spectrogram of target classes.**

The original automated approach [57] targeted chimpanzees, *Pan troglodytes* ssp. *verus*, as well as three other primate species. The present study focuses exclusively

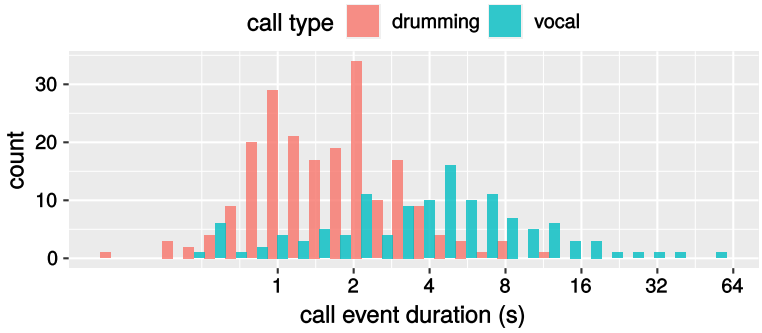

Figure 7.2: **Histogram of call type durations.**

on chimpanzees. Heinicke et al. [57] defined two chimpanzee call types for detection: (1) **drumming**, which is produced by chimpanzees when they repeatedly hit buttress roots of trees with their hands and feet, and (2) **vocalizations**, referring primarily to chimpanzee pant-hoots and screams for long-distance communication. Drumming is characterized by short energy bursts with low frequency. Vocalizations in the data set are characterized by harmonic patterns with an estimated frequency range of 200 – 2000 Hz. Fig. 7.1 shows an example spectrogram excerpt with both call types. For the remainder of this chapter, I refer to call types as *classes* and call instances as *events* in accordance with the vocabulary established in general ASR.

To construct sets for training and validation of the automated system, Kalan et al. [74, 75] sampled the data set and annotated the sampled recordings. Table 7.1 summarizes the data sets used in this study:

- The **complete test set** corresponds to the original test set constructed by Heinicke et al. [57]. They randomly sampled 358 recordings (each with a duration of 30 min) from the data pool, balanced across ARUs (one file per ARU per week) and time of day. This procedure ensured that the test set reflected diverse acoustic conditions for varying seasons, daytimes and sampling sites.
- I additionally constructed a **reduced test set** from the complete test set. It comprised all recordings from the complete test set with at least one chimpanzee event, i.e. it is a subset of the complete test set. I used this test set for repeated evaluation runs, as the complete test set was computationally expensive due to its size. The experimental setup in section 7.2.9 describes the purpose of this set in more detail.
- The **training set** contained 44 additional recordings that were likewise randomly sampled (i.e. training and test set comprised of distinct recordings). Each individual recording in the training and test set was 30 min long.
- The **validation data set** contained 25 additional recordings collected during a pilot study at the same location in 2010. Contrary to the other sets, recordings in the validation set had varying lengths with a mean duration of 1.3 min.

Two trained experts for primate vocalizations annotated call events in these recordings with precise start- and end times [57].

A central characteristic of the data set is the rarity of the target classes. The complete test set with approximately one week of recording time merely contained a total of 2.5 min of drumming and 7 min of vocalization events. The relative amount for drumming and vocalizations was 0.02 % and 0.06 %, respectively. This imbalance is representative of the real-world prevalence of chimpanzee calls obtained using PAM in natural settings. Even the reduced test set, which biased the class distribution in favor of the target classes, contained 0.16 % and 0.47 % of drumming and vocalization events, respectively. The training set contained 0.2 % and 0.35 % of drumming and vocalization events, respectively.

According to Weiss [183], the imbalance between the number of positive class examples (i.e. target calls) and negative class examples (i.e. background samples) has two effects:

- **Absolute rarity**, i.e. low amounts of training examples for the target classes. This causes classifiers to overfit individual examples rather than learning generalized patterns for the target classes, particularly in deep learning systems [46, 176].
- **Relative imbalance** between background class and target class examples. This usually induces a bias into the classifier to favor the majority class, while the minority class often is of greater interest to the user [53, 72, 91, 183]. However, the magnitude of the negative effect depends on the complexity of the classification task at hand. The algorithm might be completely unaffected if classes are linearly separable [71, 72].

		test complete	test reduced	training	validation
ARU record.	# recordings	358	50	44	25
	total duration	179 h	25 h	22 h	0.7 h
	% recordings with at least 1 chimp. call	14 %	100 %	50 %	76 %
drum.	# events	100	100	78	29
	total duration	149 s	149 s	159 s	96 s
	mean duration	1.29 s	96 s	1.76 s	2 s
vocal.	# events	50	50	53	23
	total duration	431 s	431 s	270 s	88 s
	mean duration	5.72 s	5.72 s	4.35 s	3 s

Table 7.1: **Data set overview.**

7.2.2 Detection pipeline surrounding the neural network

I modeled the task as a polyphone sound detection tasks (see foundations section 2.4.1). Figure 7.3 gives an overview of the detection pipeline at training and test time. The pipeline comprises a series of **pipeline stages** that progressively process an input audio signal (i.e. ARU recording).

The general approach and stage arrangement of *feature extraction*, *segmentation*, *CRNN*, *output concatenation* and *output thresholding* originated from Cakır et al. [22]. I further added the stages *spectrogram denoising* and *resampling*. These stages, together with the choice of the loss function, were the three components aimed at compensating class imbalance investigated in this study.

This pipeline was more elaborate than the general framework for sound prediction presented in the foundations (see section 2.4.3). In addition to that framework, the pipeline of this chapter contains several additional stages and the network outputs predictions for each spectrogram frame, instead of the entire input segment.

Although Cakir's pipeline aimed at direct polyphonic detection, the network in this study was trained for binary detection of one class at a time, i.e. drumming vs. background or vocalization vs. background. This restriction was imposed to study the effects for both classes separately.

The pipeline stages function as follows:

1. **Input**: Input are ARU recordings $\boldsymbol{x}^{(i)} \in \mathbb{R}^{L_{\text{sig}}^{(i)}}$ of length $L_{\text{sig}}^{(i)} \in \mathbb{N}$ as time-domain audio signals, where (i) is the signal index. Signals are converted to mono, normalized to a peak amplitude of 1, and kept at a sampling rate of 16 kHz. Ground truth annotations are tables that list occurrences of target calls with their start- and end times within signals.
2. **Feature map conversion**: As in study (B) and (C), I employed log-mel scaled spectrograms. The calculation procedure was equal to study (B) (see study (B) section 5.2.2). I adapted the frame length of 40 ms and hop-length of 20 ms from Cakır et al. [22]. The number of Mel-bands was 80, i.e. I doubled the frequency resolution to reduce the loss of potentially important frequency information, particularly in view of the low signal-to-noise-ratio in this task. The feature extraction process was implemented through the python library `librosa v.0.8.` [103]. Ground truth annotations were likewise converted to binary integer-encoded target vectors $\boldsymbol{y}^{(i)} \in \{0,1\}^{T^{(i)}}$, where $y_t^{(i)} = 1$ encodes presence and 0 encodes absence of the target class in time frame t of signal (i) (alias *positive* and *negative* example).
3. **Spectrogram denoising**: This stage applies a series of denoising functions to spectrograms $f_{\text{den}}(\boldsymbol{M}^{(i)}) = \boldsymbol{M}^{'(i)}$, where $\boldsymbol{M}^{(i)} \in \mathbb{R}^{T^{(i)} \times F_{\text{in}}}$ is an input spectrogram and $\boldsymbol{M}^{(i)} \in \mathbb{R}^{T^{(i)} \times F_{\text{den}}}$ is the denoised spectrogram. While the time axis size $T^{(i)}$ of each signal (i) remains unchanged, the frequency axis might be reduced to F_{den}. This stage also z-standardizes all spectrograms through global statistics calculated on the training set.

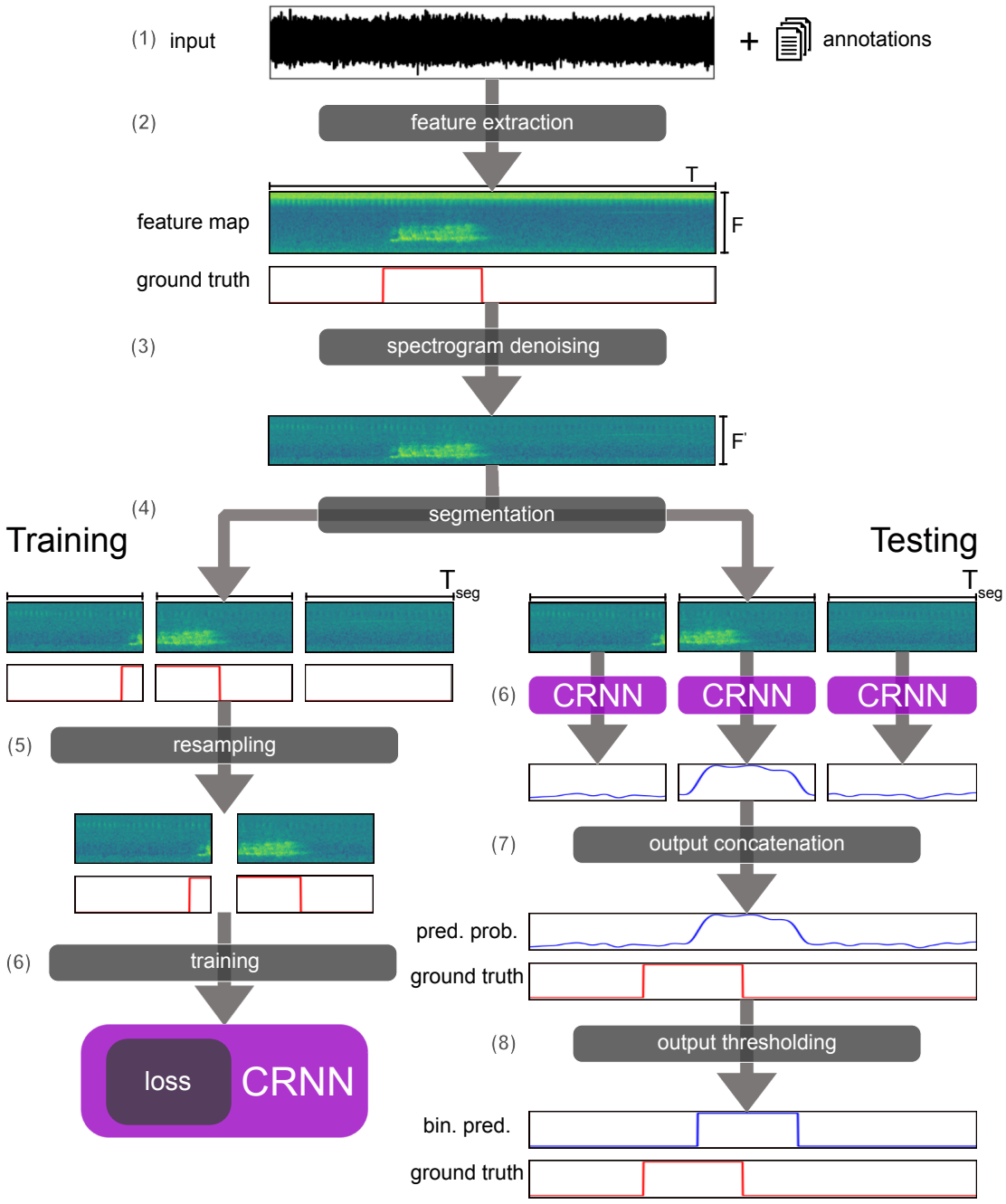

Figure 7.3: **Overview of the detection pipeline at training and test time.** In this example there is only one input signal $\boldsymbol{x}$, so that the index i is omitted.

4. **Segmentation**: In this study, I used a fixed segment length and hop length of $T_{\text{seg}} \hat{=} 10\,\text{s}$ as a balance between computational efficiency and providing sufficient temporal context in spectrogram segments for recognition of target classes. For signals with $T^{(i)} \bmod T_{\text{seg}} \neq 0$ (some signals in the validation set), the last segment was partially overlapped with the penultimate to cover $T^{(i)}$ completely if the overlap was $< 75\,\%$, or discarded otherwise. A feature map is represented as a list $\{\dot{\boldsymbol{M}}^{i,s}\}_{s=1,\dots,S^{(i)}}$, where s is the segment index.
5. **Resampling**: This stage alters the distribution between positive and negative examples through over and undersampling of segments. This stage is only active for training to not affect class distribution when validating the system. Section 7.2.6 provides details on the resampling procedure.
6. **CRNN prediction / training**: The CRNN predicts spectrogram segments to produce corresponding class probability vectors $f_{\text{crnn}}(\dot{\boldsymbol{M}}^{(i,j)}) = \hat{\boldsymbol{p}}_{i,j} \in [0,1]^{T_{\text{seg}}}$, i.e. $\hat{p}_t^{(i,j)}$ indicates the probability of the target class being present in frame t of segment j for signal i. Section 7.2.4 provides details on the CRNN configuration. Section 7.2.5 describes the loss functions experimented with.
7. **Output concatenation**: The stage concatenates the prediction segments to produce one target vector per input spectrogram $[\hat{\boldsymbol{p}}^{(i,0)}, \hat{\boldsymbol{p}}^{(i,1)}, \dots] = \hat{\boldsymbol{p}}^{(i)} \in \mathbb{R}^{T^{(i)}}$.
8. **Output binarization**: This stage binarizes predicted probabilities through thresholding: $\hat{y}_t^{(i)} = 1$, if $\hat{p}_t^{(i)} > C$, else 0. I used the unbiased threshold $C = 0.5$ (see foundations section 2.2.3 on binarization).

7.2.3 Spectrogram denoising

Spectrogram denoising is a common preprocessing step in automatic animal call detection. The aim is to increase the signal-to-noise-ratio, as recordings usually carry high amounts of noise due to the nature of open field settings. In this context, the signal is the target call of interest. Noise refers to *background sounds*, i.e. geophony (environmental sounds such as wind and rain), anthrophony (noise generated by humans, such as traffic) and biophony (sounds of animals not of interest). In the context of this study, I employed spectrogram denoising as a method for combating absolute rarity of target classes. Target event examples are present for only few background noise conditions, possibly causing the classifier to infer false coupling between noise conditions and event probability. Prior elimination of variability between noise conditions can mitigate this effect. [61, 98, 186]

Among the multitude of available methods, I chose to evaluate: (1) frequency removal and (2) spectral subtraction, as they are among the most prevalent and straight-forward methods in automatic animal call detection [61, 98, 186]. Both methods exploit the observation that background noise is fairly consistent over large periods of time, while target classes are comparatively short and seldom.

7.2.3.1 Frequency removal

Animal target calls usually occupy narrow frequency ranges. Therefore, many systems for call detection apply preprocessing filters to remove unneeded frequency ranges [53]. I calculated frequency ranges for target classes through the following proposed method:

The goal is to calculate a **class mask** $\boldsymbol{r} \in \mathbb{R}^F$, whose values r_f indicates the strength of association between Mel frequency bins f and the target class. Let $T^{(e)}_{\text{start}} \in \mathbb{N}$ and $T^{(e)}_{\text{end}} \in \mathbb{N}$ be the start and end time of event $e \in \{0, \dots, E-1\}$ indicated as spectrogram time frame indices. $\boldsymbol{M}^{(e)} \in \mathbb{R}^{T \times F}$ and $\boldsymbol{y}^{(e)}$ are the spectrogram and ground truth vectors that contain event e at some points. $\tilde{\boldsymbol{t}}^{(e)} \in [T^{(e)}_{\text{start}} - T_c, \dots, T^{(e)}_{\text{end}} + T_c]$ is the list of time frames for event e, padded by a fixed context size T_c. Then, $\tilde{\boldsymbol{M}}^{(e)}$ and $\tilde{\boldsymbol{y}}^{(e)}$ are the spectrogram and ground truth vector patches corresponding to $\tilde{\boldsymbol{t}}^{(e)}$, i.e. $\tilde{\boldsymbol{M}}^{(e)} = \boldsymbol{M}^{(e)}_{\tilde{\boldsymbol{t}}^{(e)},:}$.

For each event (e), we obtain an **event mask** $\tilde{\boldsymbol{r}}^{(e)} \in \mathbb{R}^F$ by calculating the Pearson correlation between the Mel frequency bin f and the target vector as:

$$\tilde{\boldsymbol{r}}^{(e)}_f = \frac{\operatorname{cov}(\tilde{\boldsymbol{M}}^{(e)}_{:,f}, \tilde{\boldsymbol{y}}^{(e)}_{:})}{\sigma(\tilde{\boldsymbol{M}}^{(e)}_{:,f}) \cdot \sigma(\tilde{\boldsymbol{y}}^{(e)}_{:})} \tag{7.1}$$

The total class mask $\boldsymbol{r}$ is the average of all E event masks:

$$\boldsymbol{r}_f = \frac{1}{E} \sum_{e=0}^{E-1} \boldsymbol{r}^{(e)}_f \tag{7.2}$$

The intuition behind this calculation method is as follows: Any call event causes an increase in energy of its associated frequency bands relative to the background noise. If an event is surrounded by time invariant noise, this gain is measurable as a positive correlation between a frequency band's energy and the target vector. The class mask value range is $-1 \leq \boldsymbol{r}_f \leq +1$, where $+1/-1$ indicates perfect association of a frequency band to the target vector and 0 indicates no association. Positive correlations > 0 are attributed to the actual target class acoustic content, i.e. energies active during target events. Negative correlations < 0 indicate systematic absence of band energies during events, which might be caused by other sounds commonly preceding, succeeding or pausing during target events. This approach is applicable for calculating frequency ranges of arbitrary sound classes, if the mentioned preconditions are met.

Figure 7.4 shows the class masks for both target classes. These masks were calculated exclusively on events from the training and validation set. The context length was $T_c \hat{=} 1\,\text{min}$, which roughly corresponds to the length of the longest call event in the database. When applying frequency removal, I set a correlation threshold C_r to remove frequency bands with $\boldsymbol{r}_f < C_r$. I chose $C_r = 0.025$ as a rather low threshold to ensure no significant target class information got lost. I exclusively considered positive correlations as only those were indicative of the actual target class' signal content. Negative correlations, particularly $> 2500\,\text{Hz}$, were attributed to variations in the background noise conditions such as periodic insect calls, which I aimed to eliminate in this preprocessing step.

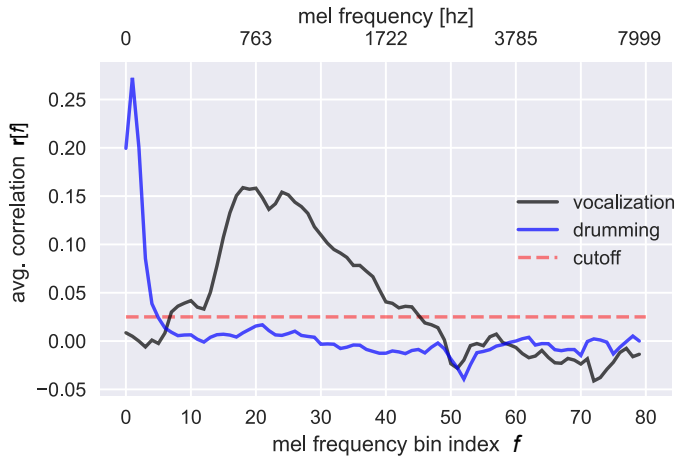

Figure 7.4: **Class masks.** Graphs indicate the strength of association between Mel frequency bands and target classes. $r_f = 0$ indicates no association and $r_f = +1/-1$ indicates perfect association. The red line indicates the threshold under which I removed frequency bands when applying frequency removal.

This procedure retained the lowest 5 frequency bins for drumming (range 0 – 152 Hz) and 39 frequency bins for vocalization (range 267 – 2097 Hz). Those frequency ranges correspond to estimations of previous studies [57]. Figure 7.5 visualizes the effect of frequency removal.

7.2.3.2 Spectral subtraction

Spectral subtraction removes background noise by subtracting a noise profile from signals. If the noise profile is assumed invariant across time/recordings, the profile is commonly estimated by averaging each frequency bin across all time steps inside a background noise region [186]. However, I observed that noise conditions vary strongly between recordings due to time of day, season and ARU location, but are fairly consistent within recordings. Consequently, I estimated local noise profiles for recordings as follows:

A spectrogram $\boldsymbol{M}^{(i)}$ was segmented into non-overlapping segments $\dot{\boldsymbol{M}}^{(i,j)} \in \mathbb{R}^{T_{\text{sub}} \times F}$ of length T_{sub} similar to the segmentation step in the processing pipeline. From each segment I subtracted each frequency bin's average value:

$$\dot{\boldsymbol{M}}'^{(i,j)}_{t,f} = \dot{\boldsymbol{M}}^{(i,j)}_{t,f} - \frac{1}{T_{\text{sub}}} \sum_{t=1}^{T_{\text{sub}}} \dot{\boldsymbol{M}}^{(i,j)}_{t,f} \tag{7.3}$$

If T_{sub} is sufficiently large compared to the expected target event duration, the average frequency band energy is dominated by the noise profile rather than the target class profile. Denoised spectrogram segments were then concatenated to reconstruct the input spectrogram progression. This method was similar to the one applied by Mac Aodha

et al. [98]. I chose $T_{\text{sub}} \hat{=} 3\,\text{min}$, so that even the maximum expected call duration contains twice as much noise as the target signal. Fig. 7.5 visualizes the effect of spectral subtraction.

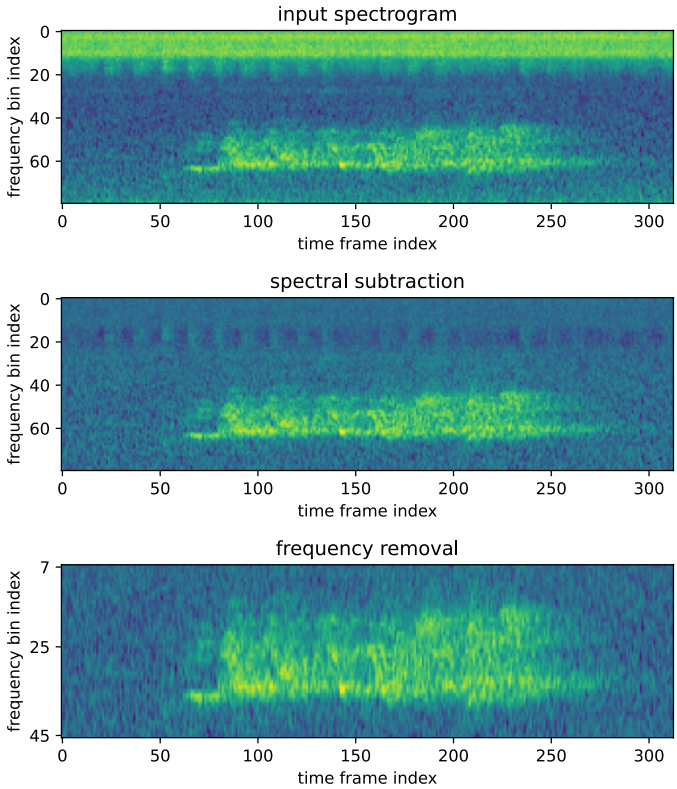

Figure 7.5: **Visualization of preprocessing operations for an example vocalization event.** Top: Input spectrogram. Middle: Effect of spectral subtraction (see section 7.2.3.2). Bottom: Effect of frequency removal (see section 7.2.3.1)

7.2.4 Convolutional recurrent neural network

As in study (B) and (C), the network architecture was subjected to a hyperparameter search. However, the search was more restricted than in the preceding studies, as investigation of the architecture influence was of secondary importance for this study.

Figure 7.6 visualizes the network architecture scheme. The scheme is based on the CRNN architecture presented in foundations section 2.3.4.4, which was originally proposed by Cakır et al. [22]. A central network property is that the temporal dimension T remains intact throughout the network, i.e. the temporal alignment between spectrogram frames and output time steps is maintained.

For further explanations on the layers and components referred to in this section, please refer to section 2.3 and particularly section 2.3.2.

7.2.4.1 Architecture scheme

The architecture contains the following stages:

1. The **input stage** appends an empty channel dimension to the spectrogram in preparation for the convolutional stage $\mathbb{R}^{T\times F} \mapsto\in \mathbb{R}^{T\times F\times(C=1)}$.
2. The **convolutional stage** corresponds to the usual pattern of convolutional stages in CNNs, i.e. alteranting convolutional and pooling layers, as described in foundations section 2.3.4.3 as well as the architecture scheme of study (C) described in section 6.2.3. However, all strides across the time axis must be 1, i.e. the time axis must not be downsampled. This ensures that the aforementioned temporal alignment between input and output frames is retained. The stage output volume is $\in \mathbb{R}^{T\times F_{\text{conv}}\times C_{\text{conv}}}$, where F_{conv} is the frequency size remaining after several pooling operations and C_{conv} is the number of filter kernels in the last convolutional layer.
3. The **frequency aggregation stage** aggregates the frequency axis and incorporates it into the channel dimension through a time-distributed function. This is analogous to the stage of the architecture scheme in study (B) (see study (B) section 5.2.3). The output volume is $\in \mathbb{R}^{T\times C_{\text{freqint}}}$.
4. The **recurrent stage** contains a recurrent layer that processes the output of the frequency aggregation stage sequentially at every time step. The output volume is $\mathbb{R}^{T\times C_{\text{rec}}}$.
5. Finally, the **output layer** consists of a time-distributed FCL $\mathbb{R}^{T\times C_{\text{rec}}} \mapsto [0,1]^T$, i.e. a single fully-connected neuron that is applied with the same weights at each time step.

The central difference between the scheme of Cakır et al. [22] and mine is the definition of the frequency aggregation stage as a generic stage, while Cakır et al. [22] defined a single fixed aggregation operation for flattening the frequency axis. This change was based on the results of study (B) and (C), which both identified the aggregation operation as an important network choice for increasing performance.

7.2.4.2 Network configuration

The following network parameters were fixed. For the convolutional stage, all convolutional layers used kernel sizes of $(5,5)$ and strides of $(1,1)$. Convolutional layers were always followed by a batch-normalization layer and ReLU activation. Pooling layers always used max pooling. The recurrent stage contained a single recurrent layer with gated recurrent units (GRU). All of these parameters were chosen in accordance to the recommendations of Cakır et al. [22]. I initialized the bias parameter of the output neuron to the respective training class distribution $\log_e(pos/neg)$, where pos and neg are the amount of positive and negative time steps, respectively. This way, networks start training with appropriate output distributions, which can prevent instability in the initial training steps, as recommended by Lin et al. [94].

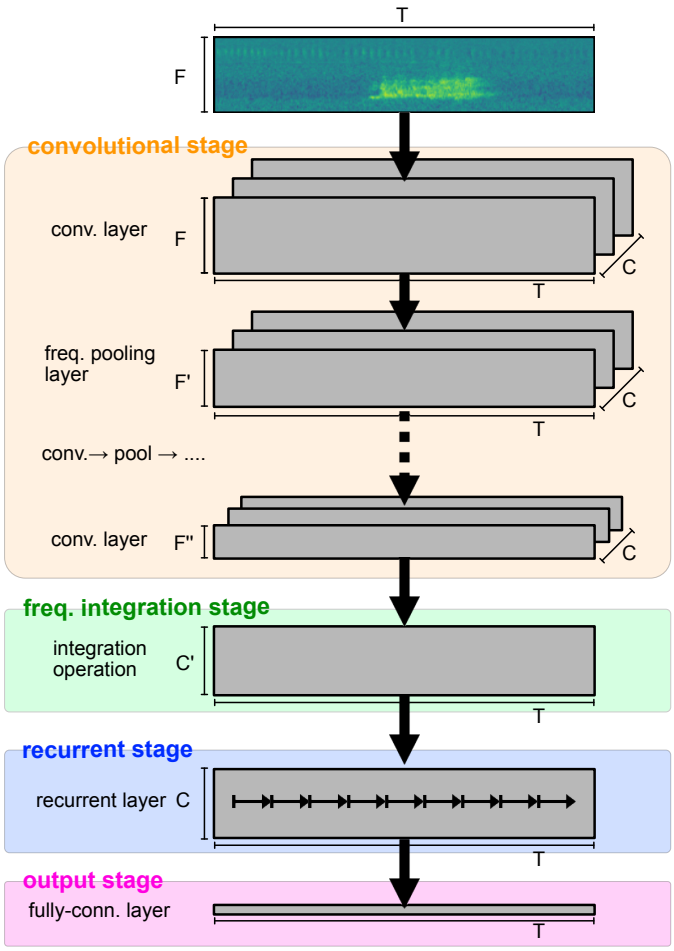

Figure 7.6: **Convolutional recurrent neural network architecture scheme**

The following network parameters were subjected to a parameter search as part of the experimental setup in this study (see section 7.2.9). The **channel size** $\in \mathbb{N}$ is the number of filters in each convolutional or recurrent layer, i.e. the number of filters/units is constant across the entire network. The **conv. stage depth** $\in \mathbb{N}$ determines the number of convolutional layers. I chose to search these parameters as they primarily determine the amount of network weights and consequently the capacity to fit the data [46]. The **pooling size** $\in \mathbb{N}$ determines the pooling size and stride for all pooling layers. The **frequency aggregation operation** is the time-distributed operation for aggregation of the frequency dimension. Finally, **recurrent bi-directional** $\in \mathbb{B}$ determined whether the recurrent layer was used bi-directionally. If yes, half of the units run forward and half backwards. I searched this parameter as I hypothesized it to be of primary importance for the network's capability to precisely locate frames of event activity.

Table 7.2 summarizes the search space. As explained further in section 7.2.9, I investigated network configuration in combination with the spectrogram denoising stage

input class	vocal		drumming	
freq. removal	no	yes	no	yes
input size $T \times F$	500×80	500×39	500×80	500×5
spectrl. subtr.	{no, yes}	{no, yes}	{no, yes}	{no, yes}
channel size	$\{32, 64, 96\}$	←	←	←
conv. stage depth	$\{2, 3, 4\}$	←	←	$\{1, 2\}$
pool size & stride	$\{2, 3, 4, 5\}$	←	←	$\{2, 3\}$
freq. integr. op.	{flatten, GAP, GMP}	←	←	←
recurrent	{no, yes}	←	←	←

Table 7.2: **Search space for network configuration and spectrogram denoising.** Table header shows input class, the top half shows denoising setting, and the bottom half shows the search space for network configuration parameters. The symbol ← indicates that a search space corresponds to the respective left cell.

setting. All search spaces were equal regardless of input class and denoising setting, except for drumming when frequency removal was applied, as this removed 93 % of the input frequency axis size.

7.2.5 Loss

The choice of loss function is among the primary algorithm-level methods for mitigating class imbalance in neural networks. There exists a multitude of loss variations aimed at compensating class imbalance. Among these, I chose to evaluate the ones that currently are most popular in classification [72]:

- **Standard BCE** as defined in the foundations section, Eq. 2.37. This is the standard loss for binary classification problems [46, chapter 6] and was also used by Cakır et al. [22].
- **Weighted BCE**, as defined in the foundations section, Eq. 2.38. Classes are associated with fixed weights according to their relative amount. This is one of the simplest and most prevalent modifications to standard BCE [72, 94, 177].
- **Focal loss (FL)** was recently proposed by Lin et al. [94] as a modification to BCE for visual object detection. Since then, it has gained wide popularity [72]. It down-weights the loss of well-classified examples irregardless of the class through $w_0 = (\hat{p})^\gamma$ and $w_1 = (1 - \hat{p})^\gamma$, where w_0 and w_1 are weights in equation 2.38. The hyperparameter γ determines the amount of down-weighting. I chose the default value $\gamma = 2$ according to study [94].
- **Weighted FL** combines FL with class weights as previously described.

stage	component	default	search space
training	loss	BCE	{weighted BCE, FL, weighted FL}
resampling	oversampling dupl. amount	2	$\{0, 2, 4, 8, 16\}$
	undersampling disc. percentage	75%	{0%, 50% 75%, 90%, 95%}

Table 7.3: **Search space for loss and resampling.**

7.2.6 Resampling

Resampling is the most prevalent data-level technique for mitigating relative class imbalance. Resampling seeks to rebalance the distribution between class examples by undersampling, i.e. discarding examples from the majority class, and oversampling, i.e. duplicating examples from the majority class. Both techniques have been successfully applied for deep learning based systems in imbalanced settings [72]. Among the set of available resampling techniques, I chose to experiment with the most prevalent and straight-forward implementations: random over and undersampling.

I implemented resampling as follows. After the segmentation step, each ARU recording was represented as a list of spectrograms $\dot{\boldsymbol{M}}^{(i,s)}$ and target vectors $\dot{\boldsymbol{y}}^{(i,s)}$ where i is the recording and s is the segment index. I separated each recording's segment list into disjoint subsets, those containing *negative* segments, i.e. segments with exclusively background time steps $Neg^{(i)} = \{\boldsymbol{M}^{(i,s)} |\ \max(\boldsymbol{y}^{(i,s)}) = 0\}_s$, and those containing *positive* segments, i.e. segments with at least one time step with a target call $Pos^{(i)} = \{\boldsymbol{M}^{(i,s)} |\ \max(\boldsymbol{y}^{(i,s)}) = 1\}_s$.

The strength of undersampling is determined through a parameter $U \in [0, 1]$ that indicates the percentage of discarded examples from $Neg^{(i)}$. The strength of oversampling is determined through a parameter $O \in \mathbb{N}$ that indicates the number of duplications of all examples in $Pos^{(i)}$, e.g. $O = 1$ means that each positive example is duplicated once. Undersampling and oversampling was performed for each input signal (i) separately. This ensured that the resulting total set of background examples displayed a certain degree of diversity even for higher discarding percentages, as noise had greater variance between than within signals.

Table 7.3 shows the default settings and search space for the resampling stage. As shown, the default setting applies some amount of over- and undersampling, while not completely balancing distributions. The reasons were (1) decreasing training time for the initial experiments by reducing the data set size, (2) ensuring that the extreme imbalance in the training set does not prevent training convergence [72], and (3) to imitate the default setting commonly used in animal call detection systems, which usually start with already undersampled databases [17, 18, 120].

7.2.7 Training setup

The optimizer was Adam with standard parameters [78] and early-stopping based on the validation set loss with a patience of 20 epochs. Training examples were shuffled between each epoch, the batch size was 64. I implemented networks and training with `tensorflow v.2.3.1` (see foundations section 2.3.3 for these terms).

7.2.8 Evaluation setup and performance metrics

This study used a fixed training-validation-testing split according to Tab. 7.1. The specific usage of the total and reduced test set depended on the experimental setup described in section 7.2.9.

I repeated each network training and evaluation 5 times and averaged performance results to reduce performance variability due to random model initializations and data set shuffling.

As evaluation metrics I chose (1) average precision (average precision) (AP), as described in foundations section 2.2.4.5 , and (2) F1 score, as described in foundation section 2.23. Both metrics are recommended for the evaluation of imbalanced class distributions [18, 34]. Metrics were implemented with `scikit-learn v.0.23`

I chose the *segment based* approach as the evaluation method, i.e. metrics were based on comparing fixed-length time intervals as evaluation instances [105]. The temporal resolutions were as follows:

- **Frame-wise resolution:** Evaluation segments corresponded directly to spectrogram frame indications $\hat{\boldsymbol{p}}$ or $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$. Consequently, the evaluation segment length was 20 ms. This resolution measures the system capacity for precise localization of events. It implicitly weighs target events according to the amount of time frames, i.e. longer events influence metric scores more than short events. This resolution was noted through the subscript AP_{frm} and $F1_{\text{frm}}$.
- **5 second resolution:** Ground truth and prediction vectors were down-sampled to 5 s intervals as evaluation instances. The down-sampling was performed through max pooling in non-overlapping segments of 5 s length. This resolution measures the system capacity for coarse localization of events. It also compensates the effect of event importance being weighted by length, as all events with length < 5 s contribute the same amount of evaluation segments. However, it also over-penalizes short false-positive-peaks. This resolution was noted through the AP_5 and $F1_5$.
- **Averaged resolution:** The average of both resolutions, e.g. $F1_{\text{avg}} = (F1_{\text{frm}} + F1_5)/2$

I used AP_{avg} as the primary evaluation metric for the following reasons: (1) It is independent of a binarization threshold, which imposes another hyperparameter that might require tuning. (2) End users for this application prefer unbinarized predictions as being more informative and (3) It summarizes the system capacity for precise and coarse event localization.

7.2.9 Experimental Setup

The central goal of the experimentation was to evaluate the influence of pipeline stages for mitigating class imbalance. Tables 7.2 and 7.3 summarize the stage's search spaces and default settings. However, the search space was too large to exhaustively evaluate through a grid search (see foundations section 2.2.2.3). Therefore, I split the experiment into two rounds to investigate local combinations of stages exhaustively.

1. **Round: Optimization of network architecture + spectrogram denoising**: For each setting of the denoising stage (no preprocessing / freq. removal / spec. sub. / freq. removal + spec. sub.) I performed a full architecture grid search according to the search space shown in Tab. 7.2. The goal was to identify the optimal combination of denoising setting and network architecture configuration. As this round performed hyperparameter selection, performance was measured on the validation set. Resampling and loss stages used their default values as shown in Tab. 7.3. The reason for optimizing these components first were as follows: (a) The network architecture is the central and only obligatory part of the pipeline and thus of primary importance for optimization. (b) I hypothesized interaction effects between denoising and network architecture parametrization, since the denoising setting altered the dimensionality and nature of the input features.
2. **Round: Optimization of loss type + resampling**: For each loss variant, I investigated each possible combination of over & undersampling according to Tab. 7.3. Network architecture and spectrogram denoising were fixed to the optimal configuration found in round 1. I investigated these stages in combination, since I suspected interaction effects, e.g. loss variants specialized in balancing stages might prefer less oversampling. For this round I used the reduced test set for evaluation, as it more accurately mirrored the real class distribution than the validation set.

Experiments were performed separately for each class.

7.3 Results

The structure of the results section follows the structure outlined in section 7.2.9. Section 7.3.1 states the analysis on the first optimization round, the architecture and spectrogram denoising. Section 7.3.2 states the analysis on the second optimization round, resampling and loss. Section 7.3.3 states the final performance evaluation.

7.3.1 Optimization of network architecture & spectrogram denoising

Figure 7.7 shows the validation set performances achieved by networks in the architecture grid search, grouped by denoising operations.

To statistically assess the influence of the investigated parameters, I constructed regression trees (see foundations section 2.2.5.2) analogous to the analysis approach of study (B) (see study (B) section 5.3). In this study I used **conditional inferences trees**

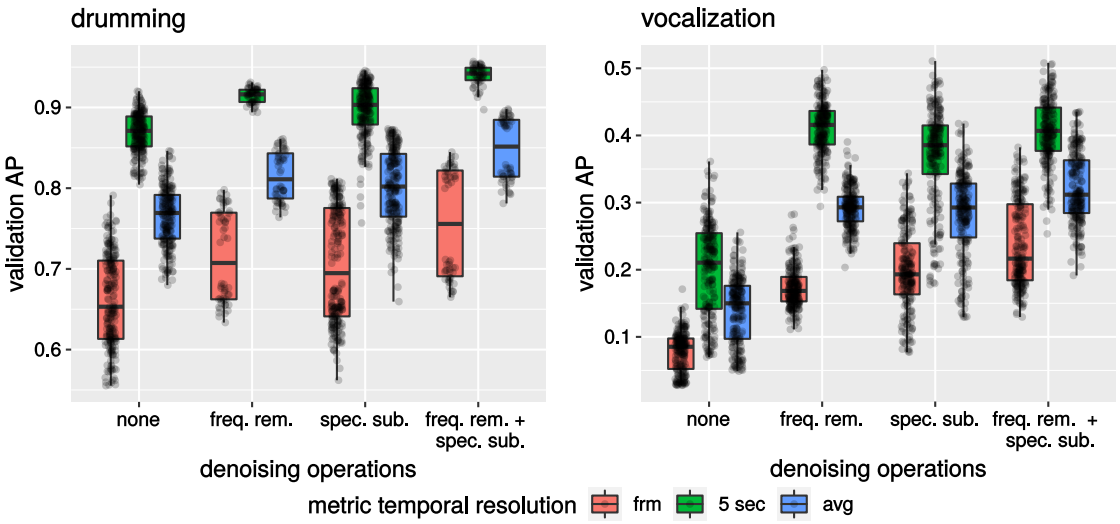

Figure 7.7: **Performances of network configurations, grouped by denoising operations.** The x-axis indicates spectrogram denoising operations, the y-axis indicates validation performance. Performance is indicated in range $[0, 1]$, e.g. $0.5 \mathrel{\hat{=}} 50\,\%$ AP. Each data point presents the performance of a network configuration produced in the architecture grid search, averaged over 5 runs.

(**c-trees**). C-trees are regression trees, where the splitting criterion is the statistical significance (p-value), as opposed to the R^2-value used by the trees of study (B). At each node, the tree chose the split criterion with the highest p-value between subgroups. This has the advantage of stating statistical association more directly.

Figure 7.10 visualizes the resulting regression trees. The target variable was the validation performance AP_{avg} and predictors were the denoising and architectural hyperparameters. The p-value-cutoff was 0.001 with Bonferroni-correction to limit trees to the most essential effects. C-trees were implemented with R-package `party v 1.4-5` [64].

I highlight the following observations based on these visualizations:

1. Detection performance was drastically higher for drumming than for vocalization, regardless of network architecture and denoising setting (drumming AP_{avg} range: $55 - 95\,\%$, vocalization: $3 - 50\,\%$). Drumming performances reached up to almost perfect scores, while the worst drumming performance was still higher than the best vocalization performance.
2. AP_5 values were generally higher than AP_{frm} values for both target classes. This means that networks were better at detecting rough localization of target events than frame-precise localization (see Fig. 7.7).
3. Both spectrogram denoising operations increased performance AP_{avg} on average (i.e. over all network configurations constructed in the grid search). Using no denoising operation yielded significantly ($p < 0.0001$) lower performance than using

either or both operations in combination. Using both operations simultaneously yielded a significantly higher ($p < 0.0001$) performance than using either or none. The performance difference between using either operation individually was not significant ($p > 0.01$).

4. Both spectrogram denoising operations increased performance regarding the maximum AP_{avg} reached by a model constructed in the architecture grid search. The order was no preprocessing $\mapsto$ freq. rem. $\mapsto$ spec. sub. $\mapsto$ freq. rem. + spec. sub. for both classes.
5. The usage of a bi-directional recurrent layer was the most important network architecture property, i.e. it increased performance for both classes with most denoising operations. For vocalization, they only increased AP_{frm} and not AP_5, i.e. they only increased the capability for precise localization of target events while rough allocation remained the same. The two clusters in Fig. 7.7 for drumming with spec. sub. + freq. rem. are explained through bi-directional layers.
6. The influence of network architecture choices decreased when applying aggregation operations. This is evidenced by the fact that both c-trees (Fig. 7.10) have the largest depths in the paths without denoising operations. The network properties with the most influence were the depth and the choice of frequency aggregation operation. However, which depth and aggregation operation optimized performance was dependent upon the denoising setting. On average, performance increased with depth and GAP for frequency aggregation.

Table 7.4 shows architectures with the highest validation performance AP_{avg} per denoising setting and their test set performances. I highlight the following observations:

1. Test performances dropped drastically compared to the validation set. This is largely due to the test set containing far more negative examples than the validation set (see Tab. 7.1).
2. While frequency removal and spectral subtraction performed similarly on the validation set, frequency removal outperformed spectral subtraction on the test set. Combining both operations yielded the highest test performance for drumming and performed on par with using frequency removal alone for vocalization.
3. For drumming, the importance of denoising functions increased on the test set. The performance difference between using no and both operations was 5 % on the validation set, but 18 % on the test set. For vocalization, denoising importance decreased on the test set. The performance difference between using none and both was 28 % on the validation set, but only 8 % on the test set.

7.3.2 Optimization of loss variant + resampling

I used the setting with the highest validation performance AP_{avg} from optimization round 1 for the experiments on loss and resampling in this second optimization round, i.e. the underlined models in Tab. 7.4.

Figure 7.8 shows the results of the experiments on the loss variant and resampling, measured on the reduced test set as AP_{avg}. Additionally, I calculated the total ratio of positive to negative segments resulting from the over / undersampling settings (positive segment = segment with at least one positive frame, see section 7.2.6). Figure 7.9 shows the corresponding c-tree analysis analogous to the one of section 7.3.1 (target: reduced test set AP_{avg}, predictors: loss variant and resampling parameters + pos/neg ratio).

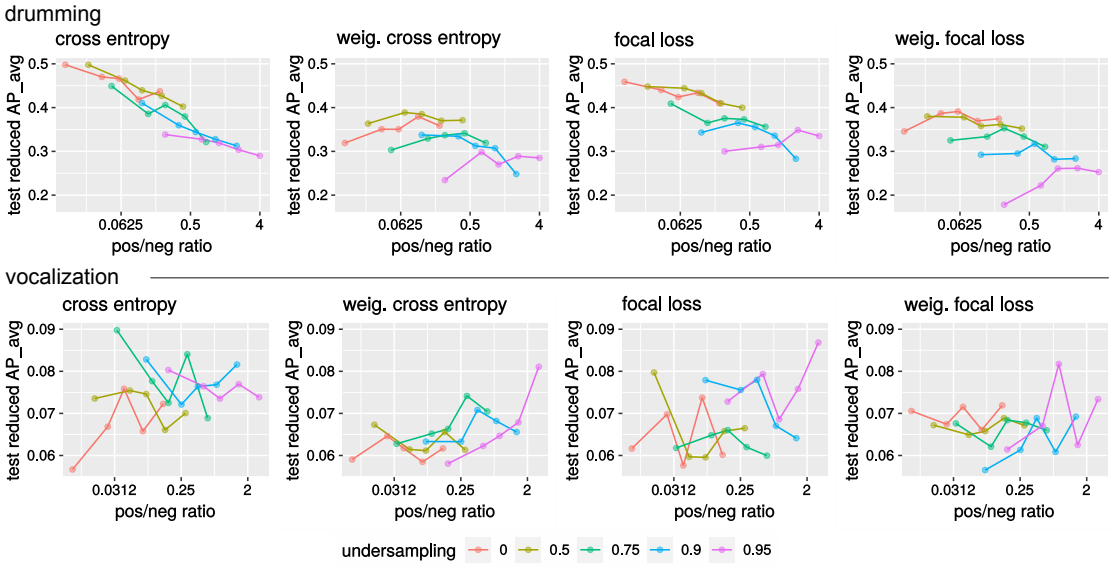

Figure 7.8: **Influence of resampling grouped by loss functions.** Upper row: drumming, lower row: vocalization. The x-axes indicate the the ratio of positive to negative segments. Oversampling is shown implicitly, where 5 data points per undersampling setting correspond to oversampling duplication amounts $\{0, 2, 4, 8, 16\}$. Performance is indicated in range $[0, 1]$, e.g. $0.5 \mathrel{\hat{=}} 50\,\%$ AP.

Loss and resampling had greater influence on drumming than vocalization, i.e. performance range was $20\,\% - 50\,\%$ AP_{avg} for drumming and $5\,\% - 9\,\%$ AP_{avg} for vocalization. For drumming I highlight the following observations:

1. Both unweighted loss functions reached higher performances than weighted functions. The global effect of weighted vs. unweighted functions was significant ($p < 0.001$). The global difference between unweighted BCE and FL was not significant. However, standard BCE had significantly higher performance than FL ($p < 0.01$) when undersampling ≤ 0.5.
2. Performance decreased globally with increased undersampling.
3. For unweighted loss functions there was a significant global association for increased performance with a decreased ratio of positive/negative segments ($p < 0.001$). This also means that performance decreased with increased oversampling.

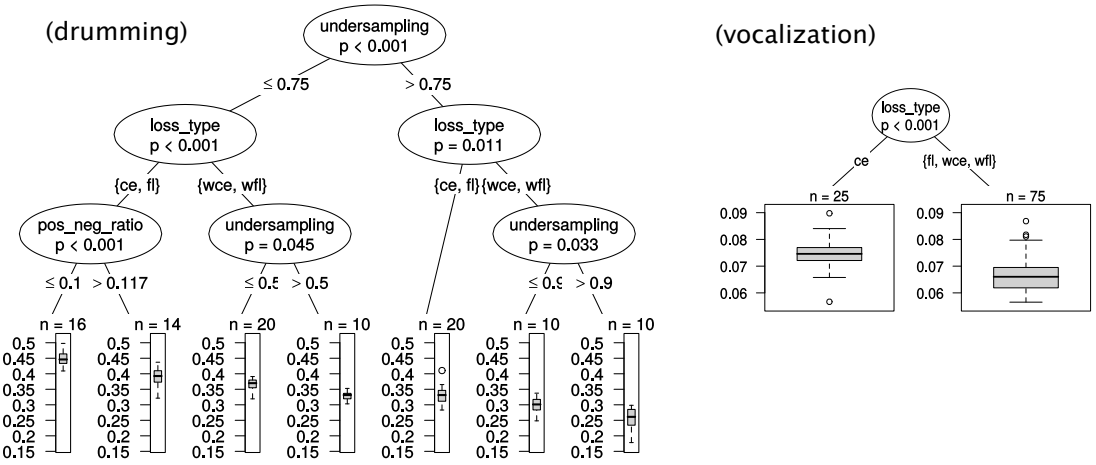

Figure 7.9: **Conditional inference trees for loss type and resampling.** Performance is indicated in range $[0, 1]$, e.g. $0.5 \mathrel{\hat{=}} 50\,\%$ AP.

4. The highest performance $49\,\%$ AP_{avg} was reached with BCE using the "raw" training database without any resampling.

For vocalization, the only significant influence was standard BCE performing better than the other loss functions, although the influence was still low in absolute terms. Resampling had no systematic influence. The highest value $AP_{\text{avg}} = 9\,\%$ was reached with no oversampling and undersampling of 0.75.

7.3.3 Final performance evaluation

Table 7.5 shows the final performance evaluation on the complete test set for the best hyperparameter settings found in this study. The denoising settings and network architecture setting correspond to the settings underlined in Tab. 7.4. The loss type and resampling correspond to the settings with the highest performance found on the reduced test set (see Fig. 7.8). AP_{avg} values dropped by approximately $60\,\%$ from the reduced to the complete test set. This is due to the sevenfold increase of negative examples compared to the complete test set, increasing the number of possible false positive predictions.

The baseline performance corresponds to the performance achieved by Heinicke et al. [57]. Their $F1$ values were computed on event-based metrics with varying event lengths based on the segmentation algorithm in their study, i.e. they are not directly comparable to these segment-based metrics. When comparing the baseline values to $F1_{\text{avg}}$-values, my performances were an improvement, increasing baseline performance. For drumming the increase was $30\,\%$ F1, which is a 7-fold increase. For vocalization, the increase was $5\,\%$ F1, which is a 25-fold increase.

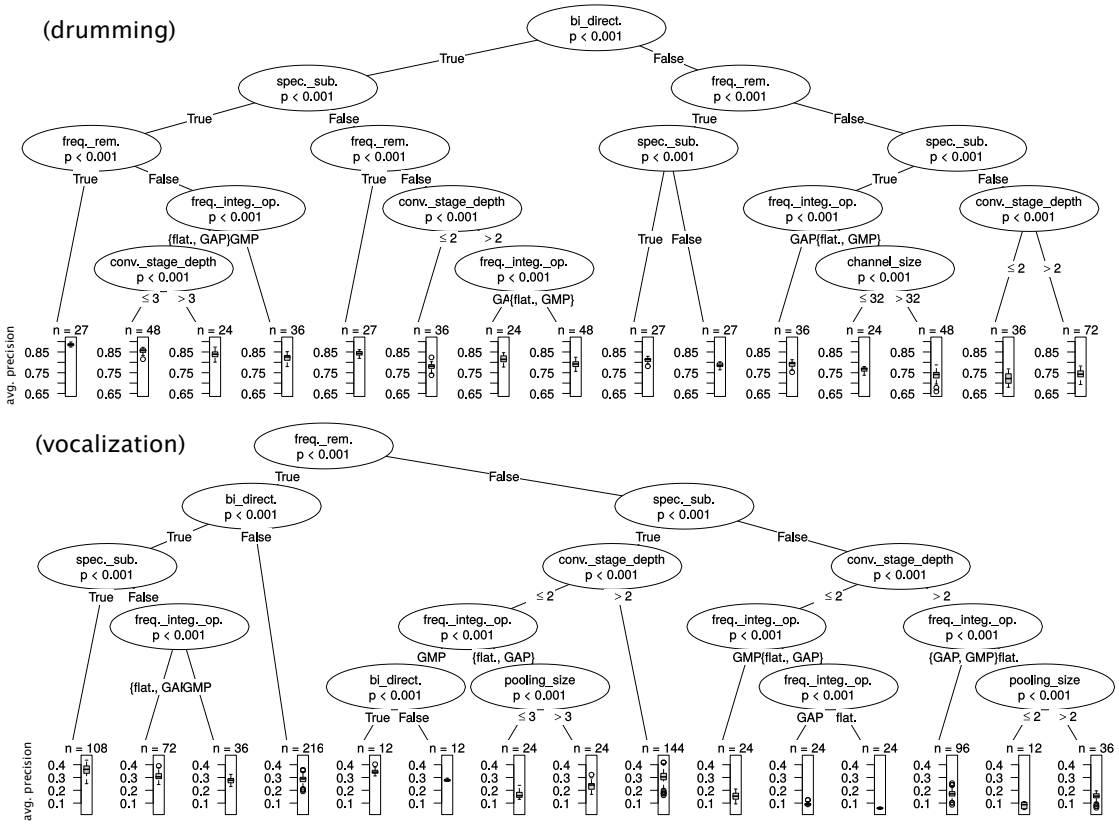

Figure 7.10: **Conditional inference trees for optimization of network architecture + denoising.** Nodes show hyperparameters with the p-value resulting from splitting the feature into groups indicated by the branches. Leafs show the performance distributions.

class	denoising setting			architectural hyperparameters					val performance [%]			complete test performance [%]		
	freq. rem.	spec. subtr.	conv. stage depth	pooling size	channel size	freq. int.	bi-direct.		AP_{frm}	$AP_{5\text{sec}}$	AP_{avg}	AP_{frm}	$AP_{5\text{sec}}$	AP_{avg}
drum.	False	False	4	2	96	GAP	True		79.1	90.1	84.6	02.5	05.9	04.2
	True	False	2	2	32	GMP	True		79.4	92.8	86.1	08.4	21.4	14.9
	False	True	4	2	96	GAP	True		80.8	93.9	87.3	04.4	07.9	06.1
	True	True	2	2	96	GAP	True		83.9	95.7	**89.8**	16.2	29.4	**22.8**
voc.	False	False	3	3	64	GAP	True		17.1	34	25.6	0.8	01.3	01
	True	False	2	3	96	GAP	True		28.3	49.8	39	01.5	0.2	**01.8**
	False	True	3	3	32	GMP	True		32.5	51.1	41.8	01.1	01.4	01.3
	True	True	2	4	96	GAP	True		38.2	48.8	**43.5**	01.6	02	**01.8**

Table 7.4: **Network architectures with highest validation performance per denoising setting.** The underlined models were selected for subsequent experiments regarding loss functions and resampling. *Complete test performance* means that the complete test set was used for testing (and not the reduced test set, see Tab. 7.1). Bold numbers highlight the highest performance reached for each class.

class	loss	resampling		pos/neg ratio train set		complete test set performance [%]						
		under-samp.	over-samp.	frm	seg	AP_{frm}	AP_5	AP_{avg}	$F1_{\text{frm}}$	$F1_5$	$F1_{\text{avg}}$	baseline $F1$ [57]
drum.	cross entr.	0	0	0.002	0.012	30.8	38.5	34.7	34	32.6	33.3	4.6
voc.	cross entr.	0.75	0	0.011	0.034	01.8	02.5	02.1	5	5.3	5.1	0.2

Table 7.5: **Final performance evaluation.**

7.4 Discussion

The F1 scores of 33 % for drumming and 5 % for vocalization might seem rather low in absolute terms. However, one has to take into account that even for humans the problem is exceptionally difficult. In addition to the rarity of the target calls, they are also very faint and subtle. This issue requires extensive training by human listeners to label calls reliably. Hence, although my results leave room for further improvement, they represent an improvement compared to previous methods.

The detection performance for chimpanzee drumming was drastically higher than for vocalization, with and without denoising/resampling. The same effect occurred in Heinicke's [57] method, although absolute values were lower. I hypothesize that the following factors contribute to the difficulty of detecting chimpanzee vocalizations: (1) Vocalizations are more complex with greater intra-class variability than drumming, as they encompass multiple call types with respect to pant hoots and screams. In comparison, drumming has a more "fixed" and stereotypical pattern. (2) The frequency bands for chimpanzee vocalization are also occupied by calls of other primate species, which are acoustically similar. However, drumming is the only animal call occupying such low frequency bands in my data set. In my experience, human listeners also have greater difficulty in identifying chimpanzee vocalizations, particularly because they confuse them for other animal calls. Thus, I reason that the vocalization class may have needed more training examples to be learned effectively by the network, given the greater difficulty of the task.

Both denoising operations, spectral subtraction and frequency removal, increased performance significantly, particularly for drumming's test set performance. This finding is in accordance with other studies on animal call detection that applied similar operations with success [61, 98, 186]. The magnitude of increase in performance by frequency removal was particularly surprising. Theoretically, networks should learn to ignore irrelevant frequency bands by themselves. I give two possible explanations for this: (1) Positive class examples were only present for few recordings. Possibly, this led the network to infer a false association between noise conditions and target call occurrence. Removing uncorrelated frequencies reduced the features that could be used for such false associations. (2) Possibly, the test set contained background noise conditions that were drastically different from the ones in the training set so that they occupy regions far away from the learned manifold and cause faulty forward-passes in the network, similar to adversarial examples [160]. I highlight that frequency removal carries the additional advantage of decreasing computation time.

I found that taking relative class balance did not increase performance. Drumming reached the highest performances using standard BCE loss without any data set resampling, i.e. using the raw, heavily imbalanced training data. Vocalization also performed best with vanilla BCE , but was insensitive to resampling. I draw the conclusion that performance-wise, combating relative class imbalance is unnecessary or even harmful. For drumming, undersampling decreased the performance regardless of the loss function, i.e. displaying diversity of the background class is important even if examples might

seem redundant for humans. Still, undersampling can reduce training time with little loss in performance if used only slightly. Drumming reached essentially the same performance using 0 % and 50 % of the training data and began dropping when only using 75 %. This finding is in contrast to other studies [53, 72, 94, 183], which usually report positive effects for balancing methods. I give the following possible explanations for this discrepancy: (1) Studies reporting positive effects of resampling commonly worked with imbalanced training sets, but balanced test sets [20, 59]. Consequently, the positive effect of resampling could be attributed to approximating class distributions between training and test set and not to compensating the imbalance within the training set. (2) Studies reporting positive effects of class weights in classification settings usually performed multi-class single-label classification with softmax activation as in multinominal logistic regression [6, 77, 181]. However, I used BCE for single-class prediction, which might be inherently more robust to class imbalances. (3) When Lin et al. [94] reported FL to outperform BCE, they performed multi-label detection for 91 classes with one network for the COCO data set. As the influence of the background class multiplicates across all positive classes in multi-label detection, loss balancing might become beneficial in such multi-label settings.

In summary, my results show that supporting the network to learn decoupling target class characteristics from background class characteristics is of primary importance for increasing performance. Spectrogram denoising explicitly supports this decoupling by discarding information from signals that are assumed to only be associated with background noise based on prior knowledge. Including more examples from the background class (no undersampling) implicitly supports this decoupling by displaying a greater amount of background noise variability to the network.

CHAPTER 8

Conclusion and Collected Discussion

8.1 Conclusion

The goal of this thesis was to advance the knowledge in automatic recognition of non-verbal acoustic communication through neural networks. The thesis focused specifically on two types of such communications, i.e. infant vocalizations and chimpanzee long-distance calls.

Computational systems for automatic recognition of sounds in general, and specifically for acoustic communication, used to be based on conventional ML approaches. In recent years, neural network based deep learning approaches emerged and demonstrated significant performance gains over conventional approaches. However, there has been little research on how to apply deep learning to non-verbal acoustic communication. There are various challenges: Particularly, deep learning usually requires large data sets, while data sets in non-verbal acoustic communication are usually small. Consequently, recognition system components require high degrees of optimization to make effective use of the available data.

The thesis comprised four studies that investigated aspects of the question: *How can we apply deep learning more effectively to automatic recognition of non-verbal acoustic communication events?*

The goal of study (A) was to investigate the assessment of infant vocalizations by laypersons to develop an infant vocalization classification scheme. Results showed that (1) acoustic classes correlate strongly with affective valence ratings, and (2) laypersons differentiate relatively few acoustic classes. Based on these findings, I proposed a classification scheme.

The goal of study (B) was to identify the neural network type with the highest performance for infant vocalization classification among the currently most prevalent ones.

Results showed that networks with convolutional stages reached the highest performance, i.e. CNNs (with and without fully-connected layers) and CRNNs. As the most important architectural choice I identified the choice of the aggregation operation for adapting volume dimensionality between network stages.

The goal of study (C) was to identify the key architectural properties of computer vision-like CNNs for automatic infant vocalization classification. Results confirmed the importance of the aggregation operation for adapting the volume dimensionality after the convolutional stage. Additionally, I identified the size of the cumulative receptive field to be a key property.

The goal of study (D) was to identify the most effective methods for combating class imbalance for automatic detection of chimpanzee calls in long-term monitoring recordings. Results showed that spectrogram denoising increased performance the most. However, methods for manupliting relative class imbalance, such as resampling or weighted loss functions, decreased performance.

8.2 Collected discussion

Studies (B), (C) and (D) all involved optimization of the neural network architecture to increase recognition performance, with varying degrees of attention to the subject. Various findings were consistent across all of these studies:

- Optimization of the network architecture always influenced performance substantially. The "wrong" architecture could not be trained at all, i.e. remained at chance-level performance. The "right" architecture managed to outperform conventional ML approaches and set new benchmark performance (if such a comparison was drawn).
- The most influential architecture choice in all studies was the aggregation operation to reduce tensor dimensionality between network stages. Consequently, when performing some form of architecture optimization, I recommend optimizing this network module first.
- There were always systematic influences of certain network properties on the performance, which were measurable through statistical methods. Consequently, applying such statistical analysis tools to rapidly identify important properties can be beneficial, as opposed to pure uninformed search such as random or grid search.
- Interaction effects always occurred in analysis of architectures. Study (C) showed this most directly, where the optimal cumulative receptive field size depended on the aggregation layer. Therefore, architecture optimization algorithms must be designed to account for such interaction effects. I applied regression trees in particular for this reason, which proved to be successful for accounting for interaction effects.

However, there were also discrepancies to be found. While the aggregation operation for reducing tensor dimensionality was highly important in all studies, the exact

chosen one to optimize the performance varied. While for study (B), the best frequency aggregation function was global max pooling, for study (C) it was global average pooling.

The implication is that, while architecture tuning is worthwhile, it must be tuned according to the task. It seems that there is not *the* best neural network architecture for *all* tasks — not even for closely related tasks in the realm of non-verbal acoustic communication. If we choose to optimize the network architecture, I recommend to focus on a few key architectural hyperparameters, namely: (1) The aggregation operation, (2) the pooling size, and (3) the depth. I highlight that these are fairly simple and straightforward network properties, which actually suggest simplifying architectures, as opposed to making them more complex.

8.3 Outlook

I recommend future research to place increasing importance on further investigation of the key, essential network properties. As of now, research in deep learning has become overly focused on increasing complexity of deep learning systems, e.g. through proposing increasingly complex architectures. However, such research fails in identifying the sources of empirical gains, if increasing complexity is done for the sake of it. An example of this trend are participant submissions in the DCASE competitions, where all participants usually propose unique network architectures with increasingly complex designs, but little justification for those choices over more pragmatic ones. Lipton and Steinhardt [95] published the paper *Research for practice: Troubling Trends in Machine Learning Scholarship*, which highlighted this very observation in all areas that apply deep learning

However, we might also strive for circumventing optimization of the network architecture entirely. First, the network architecture is merely one among many modules in a recognition system pipeline that can be optimized. We might as well operate with an unoptimized architecture, but optimize the surrounding building blocks, such as the audio representation, data augmentation etc. Second, research in ASR has begun to shift increasingly to employing pretrained networks for feature extraction. Those networks were trained on vasts amounts of data with dummy tasks, e.g. classifying youtube soundclips. An example for such an network is the *L3* network [32]. The features extracted by the convolutional stages of these networks can be used to train an conventional ML classifier. In 2020, the DCASE competition allowed the usage of pretrained networks for the first time and shifted the baseline system accordingly [58]. Although such pretrained networks might not contain the "optimal" architecture, they are trained on such vasts amount of data that it offsets such imperfection. Ironically, this approach returns to be similar to the conventional ML approach, however instead of using hand-crafted feature, we use features of a pretrained neural network.

I close this thesis with an appell:

Ultimately, systems for automatic recognition of non-verbal acoustic communication should not be developed for their own sake. The focus should be to facilitate subsequent analysis goals of practical interest, such as pain assessment in infants or home-range monitoring of chimpanzees. This requires designing systems that are applicable to real

world scenarios, i.e. long-term monitoring with imperfect acoustic conditions. However, as of now, the majority of data sets for non-verbal acoustic communication are not designed with this goal in mind. They mostly contain presegmented and preselected clips of communication events for monophone classification tasks that simulate perfect conditions, as in most challenges of the ComParE competition.

Practitioners should communicate their needs regarding computational systems for supporting their work, and support gathering data sets representative of these real-world conditions. Communication scientists should develop communication classification schemes and assessment instruments to systematize the practitioners' analysis process, and label large data sets accordingly. Finally, engineers should implement recognition systems for these real-world scenarios, rather than chasing increasing performance in benchmark data sets with little real-world value. I hope that my thesis is a small, but noticeable step in this very direction.

Bibliography

[1] Y. Abdulaziz and S. M. S. Ahmad. An accurate infant cry classification system based on continuos hidden markov model. In *2010 International Symposium on Information Technology*, volume 3, pages 1648–1652. IEEE, 2010.

[2] Y. Abdulaziz and S. M. S. Ahmad. Infant cry recognition system: A comparison of system performance based on mel frequency and linear prediction cepstral coefficients. In *Information Retrieval & Knowledge Management,(CAMP), 2010 International Conference on*, pages 260–263. IEEE, 2010.

[3] M. O. Ahmed, B. Shahriari, and M. Schmidt. Do we need "harmless" bayesian optimization and "first-order" bayesian optimization. *NIPS BayesOpt*, 2016.

[4] O. Akiyama and J. Sato. Multitask learning and semisupervised learning with noisy data for audio tagging. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2019.

[5] K. J. Anand, B. J. Stevens, P. J. McGrath, et al. *Pain in neonates and infants: pain research and clinical management series*, volume 10. Elsevier Health Sciences, 2007.

[6] R. Anand, K. G. Mehrotra, C. K. Mohan, and S. Ranka. An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969, 1993.

[7] F. Anders, M. Hlawitschka, and M. Fuchs. Automatisierte Erkennung und Klassifizierung von Säuglingslauten in Audiosignalen. In *20. Nachwuchswissenschaftlerkonferenz - Book of Abstracts*, pages 94–97. Hochschule Merseburg, 2020.

[8] F. Anders, M. Hlawitschka, and M. Fuchs. Automatic classification of infant vocalization sequences with convolutional neural networks. *Elsevier Speech Communication*, 119:36–45, 2020. https://doi.org/10.1016/j.specom.2020.03.003.

[9] F. Anders, M. Hlawitschka, and M. Fuchs. Comparison of artificial neural network types for infant vocalization classification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:54–67, 2020. https://doi.org/10.1109/TASLP.2020.3037414.

[10] F. Anders, M. Hlawitschka, and M. Fuchs. Investigation of the assessment of infant vocalizations by laypersons. *arXiv preprint*, 2021. https://arxiv.org/abs/2108.09205.

[11] F. Anders, A. K. Kalan, H. S. Kuehl, and M. Fuchs. Compensating class imbalance for acoustic chimpanzee detection with convolutional recurrent neural networks. *Ecological Informatics*, 65, 2021. https://doi.org/10.1016/j.ecoinf.2021.101423.

[12] A. Anikin, R. Bååth, and T. Persson. Human non-linguistic vocal repertoire: Call types and their meaning. *Journal of nonverbal behavior*, 42(1):53–80, 2018.

[13] A. C. Arcadi, D. Robert, and C. Boesch. Buttress drumming by wild chimpanzees: Temporal patterning, phrase integration into loud calls, and preliminary evidence for individual distinctiveness. *Primates*, 39(4):505–518, 1998.

[14] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[15] R. Barr, M. Kramer, C. Boisjoly, L. McVey-White, and I. Pless. Parental diary of infant cry and fuss behaviour. *Archives of Disease in Childhood*, 63(4):380–387, 1988.

[16] R. G. Barr. *Crying as a sign, a symptom, and a signal: Clinical, emotional and developmental aspects of infant and toddler crying.* Number 152. Cambridge University Press, 2000.

[17] C. Bergler, H. Schröter, R. X. Cheng, V. Barth, M. Weber, E. Nöth, H. Hofer, and A. Maier. Orca-spot: An automatic killer whale sound detection toolkit using deep learning. *Scientific reports*, 9(1):1–17, 2019.

[18] J. Bjorck, B. H. Rappazzo, D. Chen, R. Bernstein, P. H. Wrege, and C. P. Gomes. Automatic detection and compression for passive acoustic monitoring of the african forest elephant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 476–484, 2019.

[19] C. Boehm. Methods for isolating chimpanzee vocal communication. In *Understanding chimpanzees*, pages 38–59. Harvard University Press, 2013.

[20] M. Buda, A. Maki, and M. A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.

[21] E. H. Buder, A. S. Warlaumont, D. K. Oller, B. Peter, and A. MacLeod. An acoustic phonetic catalog of prespeech vocalizations from a developmental perspective. *Comprehensive perspectives on child speech development and disorders: Pathways from linguistic theory to clinical practice. Hauppauge, NY: NOVA*, 2013.

[22] E. Cakır, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen. Convolutional recurrent neural networks for polyphonic sound event detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(6):1291–1303, 2017.

[23] C.-Y. Chang and J.-J. Li. Application of deep learning for recognizing infant cries. In *Consumer Electronics-Taiwan (ICCE-TW), 2016 IEEE International Conference on*, pages 1–2. IEEE, 2016.

[24] C.-Y. Chang, Y.-C. Hsiao, and S.-T. Chen. Application of incremental svm learning for infant cries recognition. In *2015 18th International Conference on Network-Based Information Systems*, pages 607–610. IEEE, 2015.

[25] C.-Y. Chang, C.-W. Chang, S. Kathiravan, C. Lin, and S.-T. Chen. Dag-svm based infant cry classification system using sequential forward floating feature selection. *Multidimensional Systems and Signal Processing*, 28(3):961–976, 2017.

[26] H. Chen, Z. Liu, Z. Liu, P. Zhang, and Y. Yan. Integrating the data augmentation scheme with various classifiers for acoustic scene modeling. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2019.

[27] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014. *arXiv preprint arXiv:1412.3555*, 2014.

[28] E. Cignacco, R. Mueller, J. P. Hamers, and P. Gessler. Pain assessment in the neonate using the bernese pain scale for neonates. *Early human development*, 78 (2):125–131, 2004.

[29] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[30] R. Cohen and Y. Lavner. Infant cry analysis and detection. In *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*, pages 1–5. IEEE, 2012.

[31] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[32] J. Cramer, H.-H. Wu, J. Salamon, and J. P. Bello. Look, listen, and learn more: Design choices for deep audio embeddings. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3852–3856. IEEE, 2019.

[33] C. Cure, T. Aubin, and N. Mathevon. Sex discrimination and mate recognition by voice in the yelkouan shearwater puffinus yelkouan. *Bioacoustics*, 20(3):235–249, 2011.

[34] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.

[35] C. Dev. *Automatic Detection of Chimpanzee Vocalizations using Convolutional Neural Networks*. PhD thesis, California State University Channel Islands, 2020.

[36] J. Ebbers and R. Haeb-Umbach. Convolutional recurrent neural network and data augmentation for audio tagging with noisy labels and minimal supervision. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2019.

[37] T. Etz, H. Reetz, and C. Wegener. A classification model for infant cries with hearing impairment and unilateral cleft lip and palate. *Folia Phoniatrica et Logopaedica*, 64(5):254–261, 2012.

[38] F. Eyben, F. Weninger, F. Gross, and B. Schuller. Recent developments in opensmile, the munich open-source multimedia feature extractor. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 835–838, 2013.

[39] F. Eyben, K. R. Scherer, B. W. Schuller, J. Sundberg, E. André, C. Busso, L. Y. Devillers, J. Epps, P. Laukka, S. S. Narayanan, et al. The geneva minimalistic acoustic parameter set (gemaps) for voice research and affective computing. *IEEE transactions on affective computing*, 7(2):190–202, 2015.

[40] M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham, 2019.

[41] J. K. Ford. Acoustic behaviour of resident killer whales (orcinus orca) off vancouver island, british columbia. *Canadian Journal of Zoology*, 67(3):727–745, 1989.

[42] M. Freitag, S. Amiriparian, S. Pugachevskiy, N. Cummins, and B. Schuller. audeep: Unsupervised learning of representations from audio with deep recurrent neural networks. *The Journal of Machine Learning Research*, 18(1):6340–6344, 2017.

[43] T. Fuhr, H. Reetz, and C. Wegener. Comparison of supervised-learning models for infant cry classification/vergleich von klassifikationsmodellen zur säuglingsschreianalyse. *International Journal of Health Professions*, 2(1):4–15, 2015.

[44] S. A. Fulop. *Speech spectrum analysis*. Springer Science & Business Media, 2011.

[45] H. C. Gerhardt, F. Huber, et al. *Acoustic communication in insects and anurans: common problems and diverse solutions*. University of Chicago Press, 2002.

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

[47] C. Gorrostieta, R. Brutti, K. Taylor, A. Shapiro, J. Moran, A. Azarbayejani, and J. Kane. Attention-based sequence classification for affect detection. In *Interspeech Proceedings*, pages 506–510, 2018.

[48] G. Gosztolya, T. Grósz, and L. Tóth. General utterance-level feature extraction for classifying crying sounds, atypical & self-assessed affect and heart beats. In *Interspeech Proceedings*, pages 531–535, 2018. doi: 10.21437/Interspeech.2018-1076. URL http://dx.doi.org/10.21437/Interspeech.2018-1076.

[49] J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, pages 857–871, 1971.

[50] A. Graps. An introduction to wavelets. *IEEE computational science and engineering*, 2(2):50–61, 1995.

[51] J. A. Green, P. G. Whitney, and M. Potegal. Screaming, yelling, whining, and crying: Categorical and intensity differences in vocal expressions of anger and sadness in children's tantrums. *Emotion*, 11(5):1124, 2011.

[52] E. Gustafsson, F. Levréro, D. Reby, and N. Mathevon. Fathers are just as good as mothers at recognizing the cries of their baby. *Nature communications*, 4(1):1–6, 2013.

[53] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 73:220–239, 2017.

[54] K. A. Hallgren. Computing inter-rater reliability for observational data: an overview and tutorial. *Tutorials in quantitative methods for psychology*, 8(1):23, 2012.

[55] 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*, pages 770–778, 2016.

[56] K. He, Y. Shen, and W. Zhang. Thuee system for dcase 2019 challenge task 2. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2018.

[57] S. Heinicke, A. K. Kalan, O. J. Wagner, R. Mundry, H. Lukashevich, and H. S. Kühl. Assessing the performance of a semi-automated acoustic monitoring system for primates. *Methods in Ecology and Evolution*, 6(7):753–763, 2015.

[58] T. Heittola, A. Mesaros, and T. Virtanen. Acoustic scene classification in dcase 2020 challenge: generalization across devices and low complexity solutions. *arXiv preprint arXiv:2005.14623*, 2020.

[59] P. Hensman and D. Masko. The impact of imbalanced training data for convolutional neural networks. *Degree Project in Computer Science, KTH Royal Institute of Technology*, 2015.

[60] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, et al. CNN architectures for large-scale audio classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 131–135. IEEE, 2017.

[61] I. Himawan, M. Towsey, B. Law, and P. Roe. Deep learning techniques for koala activity detection. In *Interspeech Proceedings*, pages 2107–2111, 2018.

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

[63] S. L. Hopp, M. J. Owren, and C. S. Evans. *Animal acoustic communication: sound analysis and research methods*. Springer Science & Business Media, 1998.

[64] T. Hothorn, K. Hornik, and A. Zeileis. ctree: Conditional inference trees. *The comprehensive R archive network*, 8, 2015.

[65] 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*, pages 4700–4708, 2017.

[66] J. Huang, H. Lu, P. Lopez Meyer, H. Cordourier, and J. Del Hoyo Ontiveros. Acoustic scene classification using deep learning-based ensemble averaging. 2019.

[67] M. Huckvale. Neural network architecture that combines temporal and summative features for infant cry classification in the interspeech 2018 computational paralinguistics challenge. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH 2018*, pages 137–141. International Speech Communication Association (ISCA), 2018.

[68] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[69] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[70] I. S. James-Roberts, S. Conroy, and K. Wilsher. Bases for maternal perceptions of infant crying and colic behaviour. *Archives of disease in childhood*, 75(5):375–384, 1996.

[71] N. Japkowicz. The class imbalance problem: Significance and strategies. In *Proc. of the Int. Conf. on Artificial Intelligence*, volume 56. Citeseer, 2000.

[72] J. M. Johnson and T. M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

[73] J.-w. Jung, H.-S. Heo, H.-j. Shim, and H.-J. Yu. Knowledge distillation with specialist models in acoustic scene classification. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2019.

[74] A. K. Kalan, R. Mundry, O. J. Wagner, S. Heinicke, C. Boesch, and H. S. Kühl. Towards the automated detection and occupancy estimation of primates using passive acoustic monitoring. *Ecological Indicators*, 54:217–226, 2015.

[75] A. K. Kalan, A. K. Piel, R. Mundry, R. M. Wittig, C. Boesch, and H. S. Kühl. Passive acoustic monitoring reveals group ranging and territory use: a case study of wild chimpanzees (pan troglodytes). *Frontiers in zoology*, 13(1):1–11, 2016.

[76] A. Kershenbaum, D. T. Blumstein, M. A. Roch, Ç. Akçay, G. Backus, M. A. Bee, K. Bohn, Y. Cao, G. Carter, C. Cäsar, et al. Acoustic sequences in non-human animals: a tutorial review and prospectus. *Biological Reviews*, 91(1):13–52, 2016.

[77] G. King and L. Zeng. Logistic regression in rare events data. *Political analysis*, 9 (2):137–163, 2001.

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

[79] J. Koehler, M. Jansen, A. Rodriguez, P. J. Kok, L. F. Toledo, M. Emmrich, F. Glaw, C. F. Haddad, M.-O. Roedel, and M. Vences. The use of bioacoustics in anuran taxonomy: theory, terminology, methods and recommendations for best practice. *Zootaxa*, 4251(1):1–124, 2017.

[80] Q. Kong, T. Iqbal, Y. Xu, W. Wang, and M. D. Plumbley. Dcase 2018 challenge surrey cross-task convolutional neural network baseline. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2018.

[81] T. K. Koo and M. Y. Li. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of chiropractic medicine*, 15(2):155–163, 2016.

[82] R. Kotikalapudi and contributors. keras-vis. `https://github.com/raghakot/keras-vis`, 2017.

[83] K. Koutini, H. Eghbal-Zadeh, M. Dorfer, and G. Widmer. The receptive field as a regularizer in deep convolutional neural networks for acoustic scene classification. In *2019 27th European signal processing conference (EUSIPCO)*, pages 1–5. IEEE, 2019.

[84] K. Koutini, H. Eghbal-zadeh, and G. Widmer. Cp-jku submissions to dcase 2019: Acoustic scene classification and audio tagging with receptive-field-regularized cnns. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2019.

[85] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[86] M. Lasseck. Acoustic bird detection with deep convolutional neural networks. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE)*, 2018.

[87] Y. Lavner, R. Cohen, D. Ruinskiy, and H. IJzerman. Baby cry detection in domestic environment using deep learning. In *2016 IEEE international conference on the science of electrical engineering (ICSEE)*, pages 1–5. IEEE, 2016.

[88] J. Lawrence, D. Alcock, P. McGrath, J. Kay, S. B. MacMurray, and C. Dulberg. The development of a tool to assess neonatal pain. *Neonatal network: NN*, 12(6): 59–66, 1993.

[89] H. Le and A. Borji. What are the receptive, effective receptive, and projective fields of neurons in convolutional neural networks? *arXiv preprint arXiv:1705.07049*, 2017.

[90] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[91] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya. A survey on addressing high-class imbalance in big data. *Journal of Big Data*, 5(1):42, 2018.

[92] H. Lim, J. Park, and Y. Han. Rare sound event detection using 1d convolutional recurrent neural networks. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events Workshop (DCASE)*, pages 80–84, 2017.

[93] H.-C. Lin and J. A. Green. Infants' expressive behaviors to mothers and unfamiliar partners during face-to-face interactions from 4 to 10 months. *Infant Behavior and Development*, 32(3):275–285, 2009.

[94] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[95] Z. C. Lipton and J. Steinhardt. Research for practice: troubling trends in machine-learning scholarship. *Communications of the ACM*, 62(6):45–53, 2019.

[96] D. Luo, Y. Zou, and D. Huang. Investigation on joint representation learning for robust feature extraction in speech emotion recognition. In *Interspeech Proceedings*, pages 152–156, 2018.

[97] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning*, volume 30, page 3, 2013.

[98] O. Mac Aodha, R. Gibb, K. E. Barlow, E. Browning, M. Firman, R. Freeman, B. Harder, L. Kinsey, G. R. Mead, S. E. Newson, et al. Bat detective - deep learning tools for bat acoustic signal detection. *PLoS computational biology*, 14 (3):e1005995, 2018.

[99] A. Majid, J. S. Boster, and M. Bowerman. The cross-linguistic categorization of everyday events: A study of cutting and breaking. *Cognition*, 109(2):235–250, 2008.

[100] M. B. Manser. The acoustic structure of suricates' alarm calls varies with predator type and the level of response urgency. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1483):2315–2324, 2001.

[101] P. B. Marschik, F. B. Pokorny, R. Peharz, D. Zhang, J. O'Muircheartaigh, H. Roeyers, S. Bölte, A. J. Spittle, B. Urlesberger, B. Schuller, et al. A novel way to measure and predict development: a heuristic approach to facilitate the early detection of neurodevelopmental disorders. *Current neurology and neuroscience reports*, 17 (5):43, 2017.

[102] N. Mathevon and T. Aubin. Sound-based species-specific recognition in the blackcap sylvia atricapilla shows high tolerance to signal modifications. *Behaviour*, 138 (4):511–524, 2001.

[103] 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*, pages 18–25, 2015.

[104] P. J. McGrath, B. J. Stevens, S. M. Walker, and W. T. Zempsky. *Oxford textbook of paediatric pain*. Oxford University Press, 2013.

[105] A. Mesaros, T. Heittola, and T. Virtanen. Metrics for polyphonic sound event detection. *Applied Sciences*, 6(6):162, 2016.

[106] A. Mesaros, T. Heittola, and T. Virtanen. Tut database for acoustic scene classification and sound event detection. In *2016 24th European Signal Processing Conference (EUSIPCO)*, pages 1128–1132. IEEE, 2016.

[107] A. Mesaros, T. Heittola, E. Benetos, P. Foster, M. Lagrange, T. Virtanen, and M. D. Plumbley. Detection and classification of acoustic scenes and events: Outcome of the dcase 2016 challenge. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(2):379–393, 2017.

[108] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen. DCASE 2017 challenge setup: Tasks, datasets and baseline system. In *DCASE 2017-Workshop on Detection and Classification of Acoustic Scenes and Events*, 2017.

[109] A. Mesaros, T. Heittola, and T. Virtanen. Acoustic scene classification: an overview of DCASE 2017 challenge entries. In *2018 16th International Workshop on Acoustic Signal Enhancement (IWAENC)*, pages 411–415. IEEE, 2018.

[110] A. Mesaros, T. Heittola, and T. Virtanen. A multi-device dataset for urban acoustic scene classification. *arXiv preprint arXiv:1807.09840*, 2018.

[111] A. Mesaros, A. Diment, B. Elizalde, T. Heittola, E. Vincent, B. Raj, and T. Virtanen. Sound event detection in the DCASE 2017 challenge. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(6):992–1006, 2019.

[112] Y. Mima and K. Arakawa. Cause estimation of younger babies' cries from the frequency analyses of the voice-classification of hunger, sleepiness, and discomfort. In *2006 International Symposium on Intelligent Signal Processing and Communications*, pages 29–32. IEEE, 2006.

[113] S. Mirsamadi, E. Barsoum, and C. Zhang. Automatic speech emotion recognition using recurrent neural networks with local attention. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2227–2231. IEEE, 2017.

[114] T. M. Mitchell et al. *Machine learning*. McGraw-hill New York, 1997.

[115] G. Naithani, J. Kivinummi, T. Virtanen, O. Tammela, M. J. Peltola, and J. M. Leppänen. Automatic segmentation of infant cry signals using hidden markov models. *EURASIP Journal on Audio, Speech, and Music Processing*, 2018(1): 1–14, 2018.

[116] S. Nathani and D. K. Oller. Beyond ba-ba and gu-gu: Challenges and strategies in coding infant vocalizations. *Behavior Research Methods, Instruments, & Computers*, 33(3):321–330, 2001.

[117] S. Nathani, D. J. Ertmer, and R. E. Stark. Assessing vocal development in infants and toddlers. *Clinical linguistics & phonetics*, 20(5):351–369, 2006.

[118] H. Notman and D. Rendall. Contextual variation in chimpanzee pant hoots and its implications for referential communication. *Animal behaviour*, 70(1):177–190, 2005.

[119] S. Ntalampiras. Audio pattern recognition of baby crying sound events. *Journal of the Audio Engineering Society*, 63(5):358–369, 2015.

[120] T. Oikarinen, K. Srinivasan, O. Meisner, J. B. Hyman, S. Parmar, A. Fanucci-Kiss, R. Desimone, R. Landman, and G. Feng. Deep convolutional network for animal sound classification and source attribution using dual audio recordings. *The Journal of the Acoustical Society of America*, 145(2):654–662, 2019.

[121] D. K. Oller. *The emergence of the speech capacity*. Psychology Press, 2000.

[122] D. K. Oller, R. E. Eilers, A. R. Neal, and H. K. Schwartz. Precursors to speech in infancy: The prediction of speech and language disorders. *Journal of communication disorders*, 32(4):223–245, 1999.

[123] D. K. Oller, P. Niyogi, S. Gray, J. Richards, J. Gilkerson, D. Xu, U. Yapanel, and S. Warren. Automated vocal analysis of naturalistic recordings from children with autism, language delay, and typical development. *Proceedings of the National Academy of Sciences*, 107(30):13354–13359, 2010.

[124] D. K. Oller, E. H. Buder, H. L. Ramsdell, A. S. Warlaumont, L. Chorna, and R. Bakeman. Functional flexibility of infant vocalization and the emergence of language. *Proceedings of the National Academy of Sciences*, 110(16):6318–6323, 2013.

[125] A. Oren, A. Matzliach, R. Cohen, and H. Friedman. Cry-based detection of developmental disorders in infants. In *2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE)*, pages 1–5. IEEE, 2016.

[126] P. Pal, A. N. Iyer, and R. E. Yantorno. Emotion detection from infant facial expressions and cries. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 2, pages II–II. IEEE, 2006.

[127] M. Petroni, A. S. Malowany, C. C. Johnston, and B. J. Stevens. Classification of infant cry vocalizations using artificial neural networks (anns). In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 3475–3478. IEEE, 1995.

[128] K. J. Piczak. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2015.

[129] S. Pinker and B. MacWhinney. The bootstrapping problem in language acquisition. *Mechanisms of language acquisition*, pages 399–441, 1987.

[130] P. Podder, T. Z. Khan, M. H. Khan, and M. M. Rahman. Comparative performance analysis of hamming, hanning and blackman window. *International Journal of Computer Applications*, 96(18), 2014.

[131] F. B. Pokorny, B. Schuller, P. B. Marschik, R. Brueckner, P. Nyström, N. Cummins, S. Bölte, C. Einspieler, and T. Falck-Ytter. Earlier identification of children with autism spectrum disorder: An automatic vocalisation-based approach. In *Interspeech Proceedings*, 2017.

[132] A. Politis, A. Mesaros, S. Adavanne, T. Heittola, and T. Virtanen. Overview and evaluation of sound event localization and detection in dcase 2019. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:684–698, 2020.

[133] L. R. Rabiner and R. W. Schafer. *Introduction to digital speech processing*, volume 1. Now Publishers Inc, 2007.

[134] S. Rallabandi, B. Karki, C. Viegas, E. Nyberg, and A. W. Black. Investigating utterance level representations for detecting intent from acoustics. In *Interspeech Proceedings*, pages 516–520, 2018. doi: 10.21437/Interspeech.2018-2149. URL `http://dx.doi.org/10.21437/Interspeech.2018-2149`.

[135] B. Reichard, F. Schrump, F. Anders, C. Mönch, K. Bode, and M. Fuchs. Utilizing automatically estimated facial descriptors for pain detection during surgical interventions. In *BMT Biomedical Technology Conference VDE*, 2020.

[136] A. P. Reynolds, G. Richards, B. de la Iglesia, and V. J. Rayward-Smith. Clustering rules: a comparison of partitioning and hierarchical clustering algorithms. *Journal of Mathematical Modelling and Algorithms*, 5(4):475–504, 2006.

[137] R. L. Rodriguez and S. S. Caluya. Waah: Infants cry classification of physiological state based on audio features. In *2017 International Conference on Soft Computing, Intelligent System and Information Technology (ICSIIT)*, pages 7–10. IEEE, 2017.

[138] H. Root-Gutteridge, M. Bencsik, M. Chebli, L. K. Gentle, C. Terrell-Nield, A. Bourit, and R. W. Yarnell. Identifying individual wild eastern grey wolves (canis lupus lycaon) using fundamental frequency and amplitude of howls. *Bioacoustics*, 23(1):55–66, 2014.

[139] A. Rosales-Pérez, C. A. Reyes-García, J. A. Gonzalez, O. F. Reyes-Galaviz, H. J. Escalante, and S. Orlandi. Classifying infant cry patterns by the genetic selection of a fuzzy model. *Biomedical Signal Processing and Control*, 17:38–46, 2015.

[140] A. Rostamizadeh, A. Talwalkar, G. DeSalvo, K. Jamieson, and L. Li. Efficient hyperparameter optimization and infinitely many armed bandits. In *5th International Conference on Learning Representations*, 2017.

[141] 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.

[142] J. A. Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.

[143] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584. IEEE, 2015.

[144] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044, 2014.

[145] E. Scheiner, K. Hammerschmidt, U. Jürgens, and P. Zwirner. Acoustic analyses of developmental changes and emotional expression in the preverbal vocalizations of infants. *Journal of Voice*, 16(4):509–529, 2002.

[146] U. Schimmack and A. Grob. Dimensional models of core affect: A quantitative comparison by means of structural equation modeling. *European Journal of Personality*, 14(4):325–345, 2000.

[147] U. Schimmack and R. Rainer. Experiencing activation: Energetic arousal and tense arousal are not mixtures of valence and activation. *Emotion*, 2(4):412, 2002.

[148] B. Schuller and A. Batliner. *Computational Paralinguistics. Emotion, affect and personality in speech and language processing*. Wiley, 2014.

[149] B. Schuller, S. Steidl, A. Batliner, J. Hirschberg, J. K. Burgoon, A. Baird, A. Elkins, Y. Zhang, E. Coutinho, K. Evanini, et al. The interspeech 2016 computational paralinguistics challenge: Deception, sincerity & native language. In *17TH Annual Conference of the International Speech Communication Association (Interspeech 2016), Vols 1-5*, pages 2001–2005, 2016.

[150] B. Schuller, S. Steidl, A. Batliner, P. B. Marschik, H. Baumeister, F. Dong, S. Hantke, F. B. Pokorny, E.-M. Rathner, K. D. Bartl-Pokorny, C. Einspieler, D. Zhang, A. Baird, S. Amiriparian, K. Qian, Z. Ren, M. Schmitt, P. Tzirakis, and S. Zafeiriou. The interspeech 2018 computational paralinguistics challenge: Atypical & self-assessed affect, crying & heart beats. In *Proceedings Interspeech 2018*, pages 122–126. ISCA, 2018. doi: 10.21437/Interspeech.2018-51. URL `http://dx.doi.org/10.21437/Interspeech.2018-51`.

[151] B. Schuller, A. Batliner, C. Bergler, F. B. Pokorny, J. Krajewski, M. Cychosz, R. Vollmann, S.-D. Roelen, S. Schnieder, E. Bergelson, et al. The interspeech 2019 computational paralinguistics challenge: Styrian dialects, continuous sleepiness, baby sounds & orca activity. 2019.

[152] B. W. Schuller, A. Batliner, C. Bergler, E.-M. Messner, A. Hamilton, S. Amiriparian, A. Baird, G. Rizos, M. Schmitt, L. Stappen, et al. The interspeech 2020

computational paralinguistics challenge: Elderly emotion, breathing & masks. In *Proceedings Interspeech 2020*, pages 2042–2046. ISCA, 2020.

[153] H. Seo, J. Park, and Y. Park. Acoustic scene classification using various pre-processed features and convolutional neural networks. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2019.

[154] M. Severini, D. Ferretti, E. Principi, and S. Squartini. Automatic detection of cry sounds in neonatal intensive care units by using deep learning and acoustic scene simulation. *IEEE Access*, 7:51982–51993, 2019.

[155] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[156] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[157] R. Sisto, C. V. Bellieni, S. Perrone, and G. Buonocore. Neonatal pain analyzer: development and validation. *Medical and Biological Engineering and Computing*, 44(10):841, 2006.

[158] M. Slaney. Auditory toolbox. *Interval Research Corporation, Tech. Rep*, 10(1998), 1998.

[159] D. I. Slobin, I. Ibarretxe-Antuñano, A. Kopecka, and A. Majid. Manners of human gait: A crosslinguistic event-naming study. *Cognitive Linguistics*, 109(4):701–741, 2014.

[160] L. Smith and Y. Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.

[161] S. Smith. *Digital signal processing: a practical guide for engineers and scientists*. Elsevier, 2013.

[162] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[163] R. E. Stark. Infant vocalization: A comprehensive view. *Infant Mental Health Journal*, 2(2):118–128, 1981.

[164] R. E. Stark, S. N. Rose, and P. J. Benson. Classification of infant vocalization. *International Journal of Language & Communication Disorders*, 13(1):41–47, 1978.

[165] D. Stowell, M. D. Wood, H. Pamuła, Y. Stylianou, and H. Glotin. Automatic acoustic detection of birds through deep learning: the first bird audio detection challenge. *Methods in Ecology and Evolution*, 2018.

[166] B.-H. Su, S.-L. Yeh, M.-Y. Ko, H.-Y. Chen, S.-C. Zhong, J.-L. Li, and C.-C. Lee. Self-assessed affect recognition using fusion of attentional blstm and static acoustic features. In *Interspeech Proceedings*, pages 536–540, 2018.

[167] M. Suganuma, S. Shirakawa, and T. Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504, 2017.

[168] G. M. Sullivan and R. Feinn. Using effect size — or why the p value is not enough. *Journal of graduate medical education*, 4(3):279–282, 2012.

[169] Z. S. Syed, J. Schroeter, K. A. Sidorov, and A. D. Marshall. Computational paralinguistics: Automatic assessment of emotions, mood and behavioural state from acoustics of speech. In *Interspeech Proceedings*, pages 511–515, 2018.

[170] D. Tang, J. Zeng, and M. Li. An end-to-end deep learning framework for speech emotion recognition of atypical individuals. In *Interspeech Proceedings*, pages 162–166, 2018.

[171] T. M. Therneau, E. J. Atkinson, et al. An introduction to recursive partitioning using the rpart routines. Technical report, Mayo Foundation, 1997.

[172] 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 *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5200–5204. IEEE, 2016.

[173] M. A. T. Turan and E. Erzin. Monitoring infant's emotional cry in domestic environments using the capsule network architecture. In *Interspeech Proceedings*, pages 132–136, 2018.

[174] P. Tzirakis, S. Zafeiriou, and B. W. Schuller. End2you–the imperial toolkit for multimodal profiling by end-to-end learning. *arXiv preprint arXiv:1802.01115*, 2018.

[175] R. R. Vempada, B. S. A. Kumar, and K. S. Rao. Characterization of infant cries using spectral and prosodic features. In *2012 National Conference on Communications (NCC)*, pages 1–5. IEEE, 2012.

[176] T. Virtanen, M. D. Plumbley, and D. Ellis. *Computational analysis of sound scenes and events*. Springer, 2018.

[177] J. Wagner, D. Schiller, A. Seiderer, and E. Andre. Deep learning in paralinguistic recognition tasks: Are hand-crafted features still relevant? In *Interspeech Proceedings*, pages 147–151, 2018. doi: 10.21437/Interspeech.2018-1238. URL `http://dx.doi.org/10.21437/Interspeech.2018-1238`.

[178] N. Wahid, P. Saad, and M. Hariharan. Automatic infant cry pattern classification for a multiclass problem. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 8(9):45–52, 2016.

[179] J. Wainer and G. Cawley. Nested cross-validation when selecting classifiers is overzealous for most practical applications. *Expert Systems with Applications*, 182:115222, 2021.

[180] M. Wan, R. Wang, B. Wang, J. Bai, C. Chen, Z. Fu, J. Chen, X. Zhang, and S. Rahardja. Ciaic-asc system for dcase 2019 challenge task1. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*. Technical Report, DCASE2019 Challenge, 2019.

[181] H. Wang, Z. Cui, Y. Chen, M. Avidan, A. B. Abdallah, and A. Kronzer. Predicting hospital readmission via cost-sensitive deep learning. *IEEE/ACM transactions on computational biology and bioinformatics*, 15(6):1968–1978, 2018.

[182] A. S. Warlaumont, D. K. Oller, E. H. Buder, R. Dale, and R. Kozma. Data-driven automated acoustic analysis of human infant vocalizations using neural network tools. *The Journal of the Acoustical Society of America*, 127(4):2563–2577, 2010.

[183] G. M. Weiss. Mining with rarity: a unifying framework. *ACM Sigkdd Explorations Newsletter*, 6(1):7–19, 2004.

[184] F. Weninger, F. Eyben, B. W. Schuller, M. Mortillaro, and K. R. Scherer. On the acoustics of emotion in audio: what speech, music, and sound have in common. *Frontiers in psychology*, 4:292, 2013.

[185] R. M. Wood. Changes in cry acoustics and distress ratings while the infant is crying. *Infant and Child Development: An International Journal of Research and Practice*, 18(2):163–177, 2009.

[186] J. Xie, J. G. Colonna, and J. Zhang. Bioacoustic signal denoising: a review. *Artificial Intelligence Review*, pages 1–23, 2020.

[187] Q. Xie, R. K. Ward, and C. A. Laszlo. Automatic assessment of infants' levels-of-distress from the cry signals. *IEEE transactions on speech and audio processing*, 4(4):253, 1996.

[188] H. Yang, C. Shi, and H. Li. Acoustic scene classification using cnn ensembles and primary ambient extraction. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, 2019.

[189] X. Yao, M. Micheletti, M. Johnson, and K. de Barbaro. Classification of infant crying in real-world home environments using deep learning. *arXiv preprint arXiv:2005.07036*, 2020.

[190] A. Zabidi, L. Y. Khuan, W. Mansor, I. M. Yassin, and R. Sahak. Classification of infant cries with asphyxia using multilayer perceptron neural network. In *2010 Second International Conference on Computer Engineering and Applications*, volume 1, pages 204–208. IEEE, 2010.

[191] A. Zabidi, W. Mansor, and K. Y. Lee. Optimal feature selection technique for mel frequency cepstral coefficient feature extraction in classifying infant cry with asphyxia. *Indonesian Journal of Electrical Engineering and Computer Science*, 6 (3):646–655, 2017.

[192] G. Zamzmi, D. Goldgof, R. Kasturi, Y. Sun, and T. Ashmeade. Machine-based multimodal pain assessment tool for infants: a review. *arXiv preprint arXiv:1607.00331*, 2016.

[193] D. M. Zeifman. Acoustic features of infant crying related to intended caregiving intervention. *Infant and Child Development: An International Journal of Research and Practice*, 13(2):111–122, 2004.

[194] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[195] B. Zernikow. *Schmerztherapie bei Kindern*. Springer, 2013.

[196] Z. Zhang, A. Cristia, A. S. Warlaumont, and B. Schuller. Automated classification of children's linguistic versus non-linguistic vocalisations. *training*, 2(267):280, 2018.

[197] Z. Zhang, J. Han, K. Qian, and B. Schuller. Evolving learning for analysing mood-related infant vocalisation. In *Proc. Interspeech 2018*, pages 142–146, 2018. doi: 10.21437/Interspeech.2018-1914. URL `http://dx.doi.org/10.21437/Interspeech.2018-1914`.

[198] J. Zwarts. Semantic map geometry: Two approaches. *Linguistic Discovery*, 8(1): 377–395, 2010.

CHAPTER 9

Appendix

9.1 Appendix for study (A)

9.1.1 Starting page introductory text

Stimmung: Bitte beurteilen Sie, wie “positiv” oder “negativ” Sie die Simmungslage bzw. die Emotionalität des Säuglings auf Grundlage der gehörten Lautäußerungen einschätzen.

- Werte im linken Bereich der Skale bedeuten, dass die Stimmung des Säuglings eher negativ auf Sie wirkt. Dieser Teil der Skale erstreckt sich von “sehr negativ” über “negativ” bis “neutral".
- Werte im rechten Bereich der Skale bedeuten, dass die Stimmung des Säuglings eher positiv auf Sie wirkt. Dieser Teil der Skale erstreckt sich von “neutral” über “positiv” bis “sehr positiv".

(<here, participants were presented with a dummy-version of the scale corresponding to the one shown in fig. 4.1>)

Die horizontalen, schwarzen Linien der Skale dienen Ihnen zur Orientierung. Die Skale ist kontinuierlich, das heißt, dass alle möglichen Werte zwischen diesen Linien ebenfalls zulässig sind. Das selbe gilt für die nächsten beiden Skalen.

Wachheit: Bitte beurteilen Sie, wie hoch Sie den Grad der “Wachheit” oder “Energetisierung” des Säuglings auf Grundlage der gehörten Lautäußerungen einschätzen.

- Werte im linken Bereich der Skale bedeuten, dass der Säugling eher “schläfrig” oder “müde” oder “schlapp” auf Sie wirkt.
- Werte im rechten Bereich der Skale bedeuten, dass der Säugling eher “wach” oder “munter” oder “frisch” auf Sie wirkt.

Ruhe: Bitte beurteilen Sie, wie hoch Sie den Grad der “inneren Ruhe” oder “Gelassenheit” des Säuglings auf Grundlage der gehörten Lautäußerung einschätzen.

- Werte im linken Bereich der Skale bedeuten, dass der Säugling eher “ruhig”, “entspannt” oder “gelassen” auf Sie wirkt.
- Werte im rechten Bereich der Skale bedeuten, dass der Säugling eher innerlich “unruhig”, “angespannt”, “aufgeregt” oder “nervös” auf Sie wirkt.

Bezeichnung: Bitte wählen Sie die Bezeichnung aus, die Ihrer Meinung nach die gehörte Lautäußerung am ehesten beschreibt. Beachten Sie dabei die folgenden Punkte:

- Sie können nur eine Bezeichnung je Aufnahme auswählen. Falls Sie eine Aufnahme hören, bei der Sie mehrere Bezeichnungen als zutreffend erachten, entscheiden Sie sich bitte für diejenige, die für Sie “häufiger zu hören” oder “präsenter” ist.
- Es ist nicht erforderlich, dass Sie alle Begriffe mindestens einmal im Verlauf der Umfrage verwenden. Wenn Sie zum Beispiel denken sollten, dass zwei der gegebenen Bezeichnungen für Sie Synonyme darstellen, können Sie im Verlauf der Umfrage auch nur eine dieser beiden Bezeichnungen verwenden.

Die gegebene Liste ist nur als Ausgangspunkt zu betrachten. Sie können die Liste im Velauf der Umfrage erweitern:

- Wenn Sie der Meinung sind, dass keine der bereits gegebenen Bezeichnungen zutrifft, versuchen Sie, eine eigene Bezeichnung zu finden. Diese können Sie mit Hilfe des Punktes “Neue Bezeichnung” hinzufügen. Hinzugefügte Bezeichnungen stehen Ihnen für die restliche Umfrage dauerhaft zur Verfügung.
- Wenn Sie eine neue Bezeichnung hinzufügen, muss es sich dabei um ein Verb handeln, dass die Lautäußerungen mit einem Wort bezeichnet. Nicht zulässig sind beispielsweise zwei Worte, wie “sich aufregen”, oder nicht-Verben, wie z.b. “ängstlich".
- Falls Sie keine der gegebenen Bezeichnungen als zutreffend empfinden und Ihnen auch keine eigene, passende Bezeichnung einfällt, wählen Sie bitte die Option “Nicht klar bezeichenbar".

9.2 Appendix for study (B)

9.2.1 Complete Regression Trees

Figure 9.1 and Fig. 9.2 show the complete regression trees for all network types of the coarse and fine search process, respectively. Further details about the creation of the trees are provided in the main manuscript in sections 5.3.1 and 5.3.2. The nodes show hyperparameter names and branches show parameter values. All label names correspond to Tab. 5.2 in the main manuscript. The node labels contain prefixes *CStage*, *RStage*

or *FCStage* corresponding to the convolutional, recurrent or fully-connected stage, respectively. Leafs indicate the average performance value of their respective network configurations (validation loss in the coarse search, test UAR in the fine search). Performance values are emphasized through color-coding from red to green. The right branch of a node indicates the direction of increasing performance (decreasing loss or increasing UAR, respectively). Numbers under each leaf indicate the absolute amount and percentage of contained configurations. Visualizations were realized through the R package `rpart.plot v.3.0.8`.

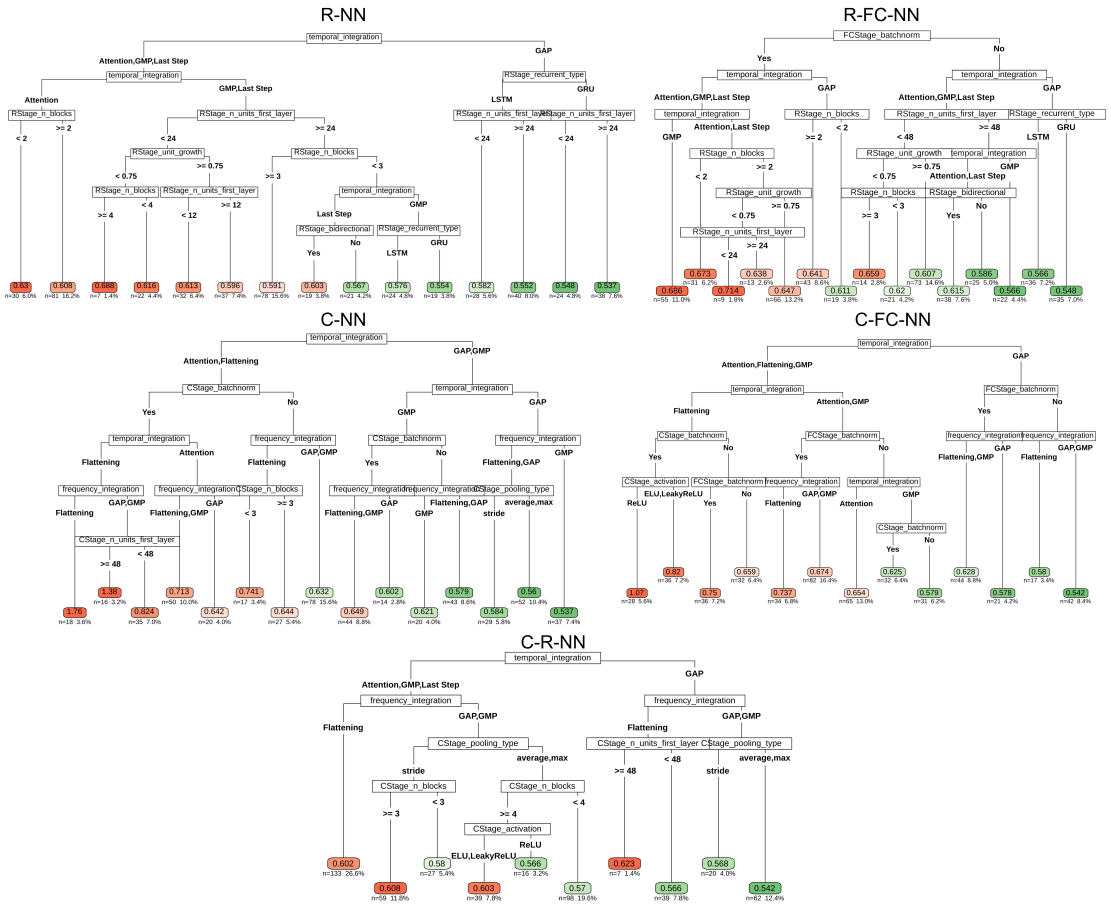

Figure 9.1: **Coarse search regression trees.**

9.2.2 Best Configurations of each Network Type

Tab. 9.1 shows each network type's configuration that reached the highest test performance, as found in the hyperparameter fine search (see section 5.3.3.2 in the main manuscript).

Table 9.1: **Best network configuration per network type.** Column, row and cell meanings correspond to Tab. II in the main manuscript.

Stage	Parameter	Network type				
		R-NN	R-FC-NN	C-NN	C-FC-NN	C-R-NN
convolutional stage	activation	-	-	ReLU	ReLU	ReLU
	n. of blocks	-	-	2	3	2
	n. units first l.	-	-	64	32	16
	unit growth	-	-	2	2	2
	batch norm.	-	-	Yes	Yes	No
	kernel shape	-	-	(1 5)	(1 5)	(1 5)
	pooling shape	-	-	(1 2)	(1 2)	(1 2)
	conv type	-	-	plain	residual	residual
	pooing type	-	-	average	average	average
	stack size	-	-	2	3	2
frequency integration	operation	Flattening	Flattening	1D GMP	1D GMP	1D GMP
recurrent stage	activation	Tanh	Tanh	-	-	-
	n. of blocks	4	2	-	-	2
	n. units first l.	64	64	-	-	1
	unit growth	0.5	2	-	-	1
	batch norm.	No	No			
	recur. type	GRU	GRU	-	-	GRU
	bidirectional	Yes	No	-	-	Yes
temporal integration	operation	1D GAP	1D GAP	1D GAP	1D GAP	1D GAP
fully-connected stage	activation	-	ReLU	-	LeakyReLU	-
	n. of blocks	-	2	-	1	-
	n. units first l.	-	1	-	2	-
	unit growth	-	1	-	1	-
	batch norm.	-	No	-	No	-
average test performance		72.89 %	72.97 %	75.22 %	75.07 %	75.35 %

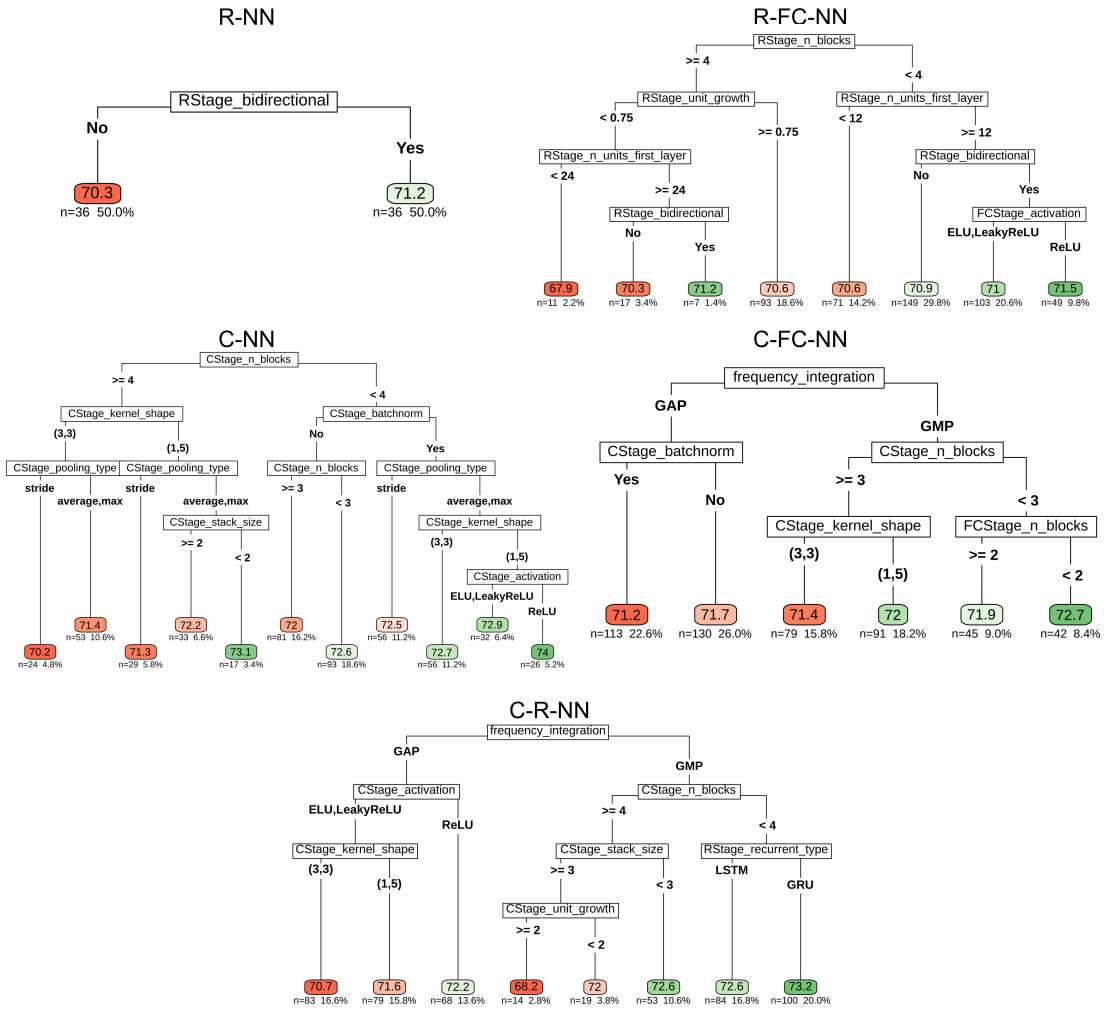

Figure 9.2: **Fine search regression trees.**

9.2.3 Visualization of Learned Features

I additionally visualized the neural network's learned feature. Figure 9.3 shows class activation maps, i.e. the input spectrograms that maximize the activations for the output neurons, respectively. Each spectrogram can be interpreted as the spectrogram template the corresponding output neuron is looking for. The visualizations were produced with the python package `keras-vis v.0.5.0` [82]. The technique follows the approach of Simonyan et al. [156]: The network is fed with a randomly initialized input, where the entries were drawn independently from a standard normal distribution. For a given output neuron, the gradient with respect to the input is computed and the input is iteratively modified to maximize the output. I run the visualization for a C-NN with 1D kernel sizes corresponding to the group of best selected hyperparameters in the study.

It can be seen that the class *neutral* focuses on high energies in the lower frequen-

cies, while *fussing* and *crying* focus on high energies in the medium and high frequencies. The templates resemble different kinds of harmonic wave patterns across the frequency axis: *neutral* resembles periodicities of both medium frequency, *fussing* a periodicity of medium frequency, and *crying* resembles a periodicity of medium and high frequencies. I hypothesize that the *neutral* activation map captures the dominance in the low frequencies, which is absent in *fussing* and *crying*. The *fussing* templates captures the evenly distributed harmonic pattern throughout the mid frequencies. *Crying* captures the narrowly stacked harmonic waves throughout the high frequencies, as well as the strong harmonic peaks throughout the mid frequencies.

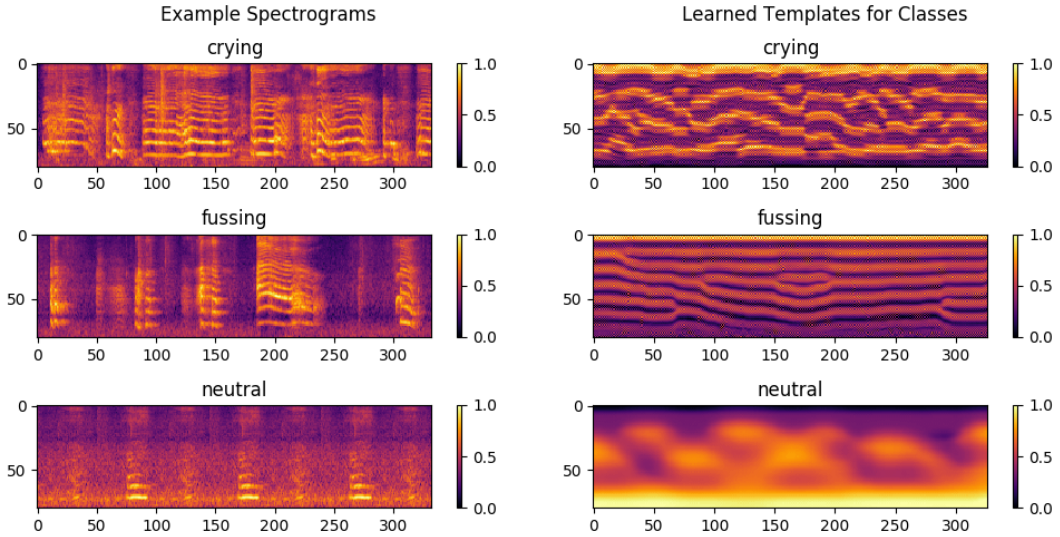

Figure 9.3: Visualization of exemplar spectrograms for classes (left) and learned class activation maps of a neural network (right). The horizontal axis represents the time dimension and the y-axis the frequency dimension. Colors encode frequency intensity.

List of Figures

2.1 Foundations: Exemplary Mel filter bank 14
2.2 Foundations: Visualization of various audio representations 15
2.3 Foundations: Overview over the various segmentation criteria for units 18
2.4 Foundations: Relationship between overfitting, underfitting and model capacity. 24
2.5 Foundations: Visualization of various evaluation procedures 30
2.6 Foundations: Early stopping in neural network training 41
2.7 Foundations: Overview over selected network architectures 42
2.8 Foundations: Schematic visualization of the major audio recognition tasks by an example 46
2.9 Foundations: Schematic visualization of the sound feature extraction process 47
2.10 Foundations: Schematic visualization of the audio recognition process 50

4.1 Study (A): Survey main window for rating of an recording 69
4.2 Study (A): Frequency of salient labels 74
4.3 Study (A): Semantic map of salient labels 76
4.4 Study (A): Relationship between aggregated label and affect ratings 76
4.5 Study (A): Semantic maps for MDS based on combinations of various rating items 79
4.6 Study (A): Other MDS plots 80
4.7 Study (A): Clustering solutions 80
4.8 Study (A): Derived classification schemes 81

5.1 Study (B): Histogram of segment durations 87
5.2 Study (B): Overview of the classification procedure 88
5.3 Study (B): Parent neural network architecture scheme 90
5.4 Study (B): Schemes of the processing stages 90
5.5 Study (B): Overall test performances of fine search sample 100
5.6 Study (B): Comparison between networks types in a contest-like setup 101
5.7 Study (B): Significances between network type performances 103

6.1 Study (C): CNN architecture scheme 110
6.2 Study (C): Histogram of system performances 115
6.3 Study (C): Confusion matrix for all audio examples 116
6.4 Study (C): PCA visualization 116

6.5 Study (C): Plots of CNN properties vs performance . . . 120

7.1 Study (D): Example spectrogram of target classes . . . 124
7.2 Study (D): Histogram of call type durations . . . 125
7.3 Study (D): Overview of the detection pipeline at training and test time . . . 128
7.4 Study (D): Class masks . . . 131
7.5 Study (D): Visualization of preprocessing operations for an example vocalization event . . . 132
7.6 Study (D): Convolutional recurrent neural network architecture scheme . . . 134
7.7 Study (D): Performances of network configurations, grouped by denoising operations . . . 139
7.8 Study (D): Influence of resampling grouped by loss functions . . . 141
7.9 Study (D): Conditional inference trees for loss type and resampling . . . 142
7.10 Study (D): Conditional inference trees for optimization of network architecture + denoising . . . 143

9.1 Study (B): Coarse search regression trees . . . 173
9.2 Study (B): Fine search regression trees . . . 175
9.3 Study (B): Visualization of learned network features . . . 176

List of Tables

4.1 Study (A): Summary of rating item perception and reliability . . . 74
4.2 Study (A): Association between label and affect ratings . . . 78

5.1 Study (B): Stage Hyperparameters . . . 92
5.2 Study (B): Search spaces of all network types . . . 94
5.3 Study (B): Best performing tree paths . . . 104

6.1 Study (C): Target classes . . . 109
6.2 Study (C): Network scheme hyperparameters and search spaces for configuration of the architecture scheme . . . 112
6.3 Study (C): Overview over data set folds . . . 113
6.4 Study (C): PCA variable loadings . . . 117
6.5 Study (C): Effects of network properties . . . 119

7.1 Study (D): Dataset overview . . . 126
7.2 Study (D): Search space for network configuration and spectrogram denoising 135
7.3 Study (D): Search space for loss and resampling . . . 136

7.4 Study (D): Network architectures with highest validation performance per denoising setting 144
7.5 Study (D): Final performance evaluation 145

9.1 Study (B): Best network configuration per network type 174

Acronyms

a.k.a. also known as. 28

ANN artificial neural network. 33

AP average precision. 29, 137

ARU autonomous recording unit. 63, 124–127, 131, 136

ASR automatic sound recognition. xiii, 3, 4, 6, 7, 9, 10, 19, 40, 41, 44, 45, 50, 53–59, 62, 107, 125, 151

BCE binary cross entropy. 39, 135, 136, 141, 142, 146, 147

CCE categorical cross entropy. 39, 96

CNN convolutional neural network. xiv, 4, 6, 8, 36, 43, 44, 56–58, 62–64, 85, 86, 91, 93, 95, 96, 105–108, 110–121, 133, 150, 177, 178

ComParE Interspeech Computational Paralinguistics Challenge. 43, 54, 55, 57, 61, 62, 86, 95–97, 100, 101, 121, 152

CP computational paralinguistics. xiii, 4, 7, 28, 48, 54–58, 63, 87, 95, 105, 113

CRNN convolutional recurrent neural network. 44, 57, 59, 62, 64, 95, 123, 124, 129, 132, 150

DCASE Detection and Classification of Acoustic Scenes and Events. 43, 54, 55, 57–59, 62, 64, 87, 95, 96, 105, 111, 151

DFT discrete Fourier transform. 11, 12

FCL fully-connected layer. 8, 34–38, 42–44, 91, 99, 106, 111, 113, 114, 117–119, 133, 150

FCNN fully-connected neural network. 42, 43, 49, 57, 62, 85

FFT fast Fourier transformation. 88

FL focal loss. 135, 136, 141, 147

GAP global average pooling. 7, 38, 94, 95, 99, 104, 105, 112, 113, 115, 117–121, 135, 140, 144, 151, 174

GMM Gaussian mixture model. 57, 62, 63

GMP global max pooling. 38, 94, 95, 99, 104, 105, 135, 144, 174

GRU gated recurrent unit. 37, 92, 94, 95, 99, 104, 106

i.i.d. independent and identically distributed. 50

IRR inter-rater reliability. 73, 74

LLD low level descriptor. 48, 56, 57, 59, 62, 106

LPCCs linear predictive cepstral coefficients. 62

LSTM long short-term memory. 37, 57, 91, 92, 94, 95, 99

MDS multidimensional scaling. 75–81, 177

MFCCs Mel frequency cepstral coefficients. 57, 62

ML machine learning. 4, 7, 9, 16, 18, 19, 22, 23, 29, 32, 33, 45, 48, 50, 53, 55, 56, 58, 62, 63, 65, 121, 123, 149–151

PAM passive acoustic monitoring. 63, 64, 123, 124, 126

PCA principal component analysis. 114–118, 177, 178

RNN recurrent neural network. 8, 43, 57, 62, 85, 95

SGD stochastic gradient descent. 40

SML supervised machine learning. xiii, 18–21, 23, 25, 27, 29, 31, 33

STFT short-time Fourier transform. 11, 12, 46, 57, 106, 109

SVM support vector machine. 57, 62

UAR unweighted average recall. 28, 73, 78, 96, 99, 113, 115, 173

USML unsupervised machine learning. 19, 72