

Masterarbeit

Angefertigt im Rahmen der Forschungstätigkeit am Fraunhofer IDMT  
vom 01.10.2014 bis 31.03.2015

**'Schallquellenlokalisierung von Sprachsignalen mittels Deep Neural Networks'**  
'Sound source localization of speech signals using Deep Neural Networks'

Gutachter/Betreuer

Prof. Dr. Stefan Weinzierl (TU Berlin)  
Stephan Gerlach M.Sc. (Fraunhofer IDMT)  
Dr.-Ing. Niko Moritz (Fraunhofer IDMT)

18. Januar 2018

verfasst von: Reinhild Roden



Technische Universität Berlin  
Fakultät I - Fachgebiet Audiokommunikation  
Einsteinufer 17c - D-10587 Berlin

Fraunhofer Institute for Digital Media Technology IDMT  
Project Group Hearing, Speech and Audio Technology  
Marie-Curie-Strasse 2 - D-26129 Oldenburg



**Fraunhofer**

au-  
dio-  
kom-  
mu-  
nika-  
tion





## Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt gegenüber der Fakultät I der Technischen Universität Berlin, dass die vorliegende, dieser Erklärung angefügte Arbeit selbstständig und nur unter Zuhilfenahme der im Literaturverzeichnis genannten Quellen und Hilfsmittel angefertigt wurde. Alle Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind kenntlich gemacht. Ich reiche die Arbeit erstmals als Prüfungsleistung ein. Ich versichere, dass diese Arbeit oder wesentliche Teile dieser Arbeit nicht bereits dem Leistungserwerb in einer anderen Lehrveranstaltung zugrunde lagen.

Mit meiner Unterschrift bestätige ich, dass ich über fachübliche Zitierregeln unterrichtet worden bin und verstanden habe. Die im betroffenen Fachgebiet üblichen Zitiervorschriften sind eingehalten worden.

Eine Überprüfung der Arbeit auf Plagiate mithilfe elektronischer Hilfsmittel darf vorgenommen werden.

Berlin, den 18. Januar 2018

.....  
Reinhild Roden





## **Danksagung**

Herrn Prof. Stefan Weinzierl, Stefan Götze und vor allem Stephan Gerlach und Dr. Ing. Niko Moritz möchte ich für die fachliche Unterstützung durch ihre große Expertise danken; Professoren, Mitarbeitern und Kommilitonen des Fachgebiets der Audiokommunikation und -technologie möchte ich für eine interessante, lehrreiche und schöne Studienzeit in Berlin danken, sowie dem Fachgebiet dem Fraunhofer Institut für Digitale Medientechnologie IDMT in Oldenburg für die freundliche Zusammenarbeit in der Abschlussphase des Studiums.

Ein besonderer Dank gilt meiner Familie.



## **Zusammenfassung**

In den letzten Jahren wurden künstliche neuronale Netze vor Allem im Bereich der automatischen Spracherkennung erfolgreich eingesetzt. Als Informationen verarbeitende Systeme können neuronale Netze z.B. mittels Backpropagation-Algorithmus oder aber sog. Restricted Boltzmann Maschinen trainiert werden, um Muster am Eingang zu erkennen. Ebenso ist es möglich, dass neuronale Netze Schallquellen lokalisieren lernen. Einige Studien beschränken sich dabei auf Winkel der Horizontalebene ohne den Einbezug vertikaler Auslenkungen. Diese Arbeit zeigt die Implementierung eines tiefen neuronalen Netzwerkes (DNN) für die sphärische Schallquellenlokalisierung unter Verwendung von Backpropagation und Bottleneck-Merkmalen. Aus Sprach- und Rauschsignalen, die zunächst mit kopfbezogenen Impulsantworten (HRIR) gefaltet worden sind, werden für den Eingang des neuronalen Netzes Merkmale extrahiert und auf ihre Eignung untersucht. Zudem wurden unterschiedliche Arten des DNN-Ausgangs betrachtet. Letztlich konnten passende Netzkonditionen und mögliche Ein- und Ausgangsvektoren gefunden werden. Durch das neuronale Netz geschätzte Richtungen eines frontalen und seitlichen Fensterausschnitts (mit je 49 Richtungen) zeigten sehr gute Übereinstimmungen mit zu erkennenden Richtungen besonders für Mel-Frequenz-Cepstrum-Koeffizienten.

**STICHWORTE:** sphärische Schallquellenlokalisierung; Maschinelles Lernen; Tiefe Neuronale Netze (DNN); Backpropagation; Bottleneck-Merkmale; kopfbezogene Übertragungsfunktion (HRTF);

## **Abstract**

In recent years artificial neural networks have been successfully applied especially in the context of automatic speech recognition. For example, neural networks are trained by backpropagation or restricted Boltzmann machines as information processing systems to classify patterns at the input of the system. In some previous studies using neural nets localization is limited to the estimation of horizontal plane without considering further directions. The present study shows the implementation of a deep neural network (DNN) architecture for spherical acoustic source localization using the backpropagation algorithm and bottleneck features. Head related impulse responses (HRIR) were convolved with speech and noise signals in order to achieve raw data for feature extraction. Extracted features were investigated as suitable input for the network. In addition, different kinds of DNN outputs were considered. Finally, a set of applicable parameters could be identified such as network conditions as well as input and output vectors. Estimated directions of a frontal and lateral grid (49 directions for each grid) showed very high congruence with target directions especially for mel frequency cepstral coefficients.

**KEYWORDS:** spherical sound localization; machine learning; deep neural network (DNN); backpropagation; bottleneck feature; head related transfer function (HRTF);

# Inhaltsverzeichnis

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Einleitung</b>  | <b>1</b>  |
| <b>2</b> | <b>Stand der Forschung</b>                                     | <b>4</b>  |
| 2.1      | Schalllokalisation . . . . .                                   | 4         |
| 2.2      | Kopfbezogene Übertragungsfunktionen . . . . .                  | 4         |
| 2.3      | Abhängigkeit von Azimuth und Elevation . . . . .               | 5         |
| 2.4      | Neuronale Netze . . . . .                                      | 6         |
| 2.4.1    | Bottleneck . . . . .   | 8         |
| 2.4.2    | Backpropagation als Lernverfahren . . . . .                    | 8         |
| <b>3</b> | <b>Datensatz</b>   | <b>11</b> |
| 3.1      | HRTF . . . . .   | 11        |
| 3.2      | Rauschsignale . . . . .  | 12        |
| 3.3      | Sprachsignale . . . . .  | 13        |
| 3.4      | Binaurale Signale . . . . .                                    | 13        |
| 3.5      | Ausbreitungsdämpfung zur Simulation von Entfernungen . . . . . | 13        |
| 3.6      | Merkmalsextraktion . . . . .                                   | 14        |
| 3.6.1    | ILD und ITD . . . . .  | 14        |
| 3.6.2    | Spektrale Merkmale . . . . .                                   | 15        |
| 3.6.3    | MFCC . . . . .   | 15        |
| 3.7      | Targetvektoren . . . . .                                       | 16        |
| 3.8      | Verwendete neuronale Netze . . . . .                           | 17        |
| 3.8.1    | OXlearn . . . . .  | 17        |
| 3.8.2    | Änderungen/Erweiterungen in der OXlearn-Toolbox . . . . .      | 17        |
| <b>4</b> | <b>Vorstudie für einzelne Dimensionen</b>                      | <b>19</b> |
| 4.1      | Untersuchte Merkmale . . . . .                                 | 19        |
| 4.2      | Netzkonditionen . . . . .                                      | 20        |
| 4.3      | Allgemeine Vorgehensweise der Evaluation . . . . .             | 21        |
| 4.4      | Ergebnisse und Diskussion der Vorstudie . . . . .              | 22        |
| <b>5</b> | <b>Sphärische Lokalisation</b>                                 | <b>25</b> |
| 5.1      | Netzkonditionen . . . . .                                      | 26        |
| 5.2      | Untersuchung der Dimensionsunabhängigkeit . . . . .            | 26        |
| 5.3      | Dimensionsabhängige Untersuchung . . . . .                     | 28        |
| 5.3.1    | Vergleich Rausch- und Sprachsignale . . . . .                  | 29        |
| 5.3.2    | Vergleich frontaler und seitlicher Lokalisation . . . . .      | 30        |
| 5.3.3    | Lohnt der Mehraufwand des Bottlenecks? . . . . .               | 30        |

|          |  |            |
|----------|--|------------|
| 5.3.4    | Eignung der Targetvektoren . . . . .                           | 31         |
| 5.3.5    | Auswertung zur Wahl der Merkmale . . . . .                     | 31         |
| <b>6</b> | <b>Fazit - Ausblick</b>  | <b>32</b>  |
| <b>A</b> | <b>Abbildungen</b>   | <b>40</b>  |
| i        | Confusion-Matrizen zur sphärischen Lokalisation . . . . .      | 40         |
| <b>B</b> | <b>Tabellenanhang</b>  | <b>113</b> |
| <b>C</b> | <b>Quellcode Matlab</b>  | <b>114</b> |
| i        | HRTF Processing . . . . .                                      | 114        |
| ii       | Ausbreitungsdämpfung . . . . .                                 | 124        |
| iii      | Rauschsignal . . . . .   | 126        |
| iv       | Feature Extraktion und Bildung des Ausgangsdatensatz . . . . . | 129        |
| v        | Netzwerk . . . . .   | 151        |
| vi       | Evaluation . . . . .   | 172        |
| <b>D</b> | <b>Datenträger</b>   | <b>185</b> |
| i        | example_input_data . . . . .                                   | 185        |
| a        | noise – front/side . . . . .                                   | 185        |
| b        | speech – front/side . . . . .                                  | 185        |
| ii       | HRTF . . . . .   | 185        |
| a        | HRTF_short . . . . .   | 185        |
| b        | HRTF_original_Thiemann . . . . .                               | 185        |
| iii      | logatom_used_data . . . . .                                    | 185        |
| iv       | noises . . . . .   | 185        |
| v        | sourcecode . . . . .   | 185        |
| a        | OXlearn_added . . . . .  | 185        |
| b        | OXlearn_added_files . . . . .                                  | 185        |
| c        | OXlearn1.1_original . . . . .                                  | 185        |
| d        | tools . . . . .  | 185        |
| vi       | Arbeit im PDF-Format (hyperref) . . . . .                      | 186        |
| vii      | Arbeit im PDF-Format (Leseformat) . . . . .                    | 186        |

# Abkürzungsverzeichnis

**DNN** tiefes oder mehrschichtiges neuronales Netz/Deep Neural Network

**ERB** äquivalente Rechteckbandbreite/Equivalent Rectangular Bandwidth

**HRIR** kopfbezogene Impulsantwort/Head Related Impulse Response

**HRTF** kopfbezogene Übertragungsfunktion/Head Related Transfer Function

**IDMT** Fraunhofer-Institut für digitale Medientechnologie

**ILD** interauraler Pegelunterschied/Interaural Level Difference

**IPD** interauraler Phasenunterschied/Interaural Phase Difference

**ITD** interauraler Zeitunterschied/Interaural Time Difference

**MFCC** Mel-Frequenz-Cepstrum-Koeffizienten/Mel      Frequency      Cepstral  
Coefficients

**STFT** Kurzzeit-Fourier-Transformation/Short-Term Fourier Transformation

# Kapitel 1

## Einleitung

Neuronale Netze werden insbesondere erfolgreich im Feld der automatischen Spracherkennung eingesetzt. Als informationsverarbeitende Systeme werden neuronale Netze eingesetzt, um ein Muster am Eingang des Systems zu klassifizieren. Dies kann u.a. durch den Backpropagation-Algorithmus oder die Boltzmann-Maschine beschränkter Verbindungen geschehen.

Die Lokalisation des horizontalen Schalleinfallswinkels ist z.B. nützlich für Hörhilfen und räumliche Filteralgorithmen, die den Schall einer Quelle verstärken oder dämpfen können. So können am Beispiel eines Hörgerätes nicht nur pauschal Quellen frontalen Schalleinfalls verstärkt werden, sondern gezielt gewünschte Geräusche bzw. Stimmen im Einfallswinkel detektiert und umliegende Quellen zugunsten des gewünschten Schalls gedämpft werden. Je präziser lokalisiert werden kann, desto effizienter können auf Schalllokalisation beruhende Algorithmen von Sprachsignalverarbeitung eingesetzt werden. Eine stabile dreidimensionale und bereits eine zweidimensionale Schalllokalisation könnte zu einer erheblichen Verbesserung der auf Schalllokalisation beruhenden Algorithmen beitragen. Auch ohne Sprache sind viele Anwendungen denkbar, wie etwa für die Geräuschanalyse akustischer Szenen.

Es finden sich viele Studien zur akustischen Quellenlokalisation mittels neuronalen Netz. Häufig werden auditorisch motivierte Lokalisationsparameter, wie etwa die interaurale Pegeldifferenz (interaural level difference ILD), interaurale Phasendifferenz (interaural phase difference IPD) und die interaurale Zeitdifferenz (interaural time delay ITD), siehe z.B. Datum et al. 1996 [DPM96], Czyzewski 2003 [Czy03], May et al. 2011 [MvdPK11], für eine stabile Lokalisationsleistung in horizontaler und vertikaler Schalleinfallsrichtung erfolgreich verwendet. Horizontale und vertikale Auslenkung werden beschrieben durch Azimuth  $\vartheta$  und Elevation  $\varphi$ . Neti et al. [NYS92] versuchten durch das Training eines neuronalen Netzes die Lokalisation von Katzen zu imitieren. Dazu wurden kopfbezogene Übertragungsfunktionen von Katzen am Ohrkanal gemessen und verwendet. Es wurde gezeigt, dass zur Lokalisation verwendete Merkmale dimensionsabhängig sein können und sich demnach die Azimuthinformation mit Änderung der Elevation verändert und umgekehrt.

Die auditorische Tiefenwahrnehmung ist bei Weitem nicht so genau wie das Lokalisieren einer Richtung (vgl. Blauert 2008, S. 98 f. [BB08]; Blauert 1997, S. 116-137 [Bla97]). Die Distanz ist mit auditorischen Parametern nur schwer zu

schätzen. Eine dreidimensionale Lokalisation anhand eines auditorischen Modells ist daher kaum denkbar. Viele Studien beschränken sich darüber hinaus auf eine Dimension und dabei auf die Lokalisation des Azimuthwinkels (z.B. Yuh et al. 1992 [Yuh92]).

Vorliegende Arbeit knüpft an Studien sphärischer Lokalisation an. Die frei verfügbare OXlearn-Toolbox bietet einen guten Ausgangspunkt ein neuronales Netz zu entwickeln. Innerhalb dieser werden Funktionen erweitert, um auch mehrschichtige neuronale Netze mit und ohne einem sogenannten Bottleneck trainieren und testen zu können. Für diese Arbeit programmierte DNNs sind nicht rekurrent und werden über das Gradientenverfahren der Rückwärtspropagierung bzw. Backpropagation trainiert.

In einer Vorstudie [RMG<sup>+</sup>15] werden tiefe neuronale Netze (Deep Neural Network DNN) verwendet, um die Position einer Schallquelle hauptsächlich für zwei räumliche Dimensionen mit Azimuth-  $\vartheta$  und Elevationswinkel  $\varphi$  der Horizontal- und Medianebene zu schätzen. Die Lokalisation erfolgt für einzelne Dimensionen in verschiedenen neuronalen Netzen. Es werden vorbereitend Merkmale aus den mit kopfbezogenen Übertragungsfunktionen gefalteten Sprachsignalen extrahiert, die die Richtungsinformationen enthalten. Explorativ werden geeignete Netzwerkparameter gefunden. Ziel ist eine genaue Lokalisation bei möglichst wenig Rechenaufwand, der zudem parallelisierbar sein sollte.

Des weiteren findet sich in dieser Arbeit zur Vorstudie der Versuch den Radius  $r$  über die Ausbreitungsdämpfung zu ermitteln. Gegenüber dem menschlichen Gehör ist das neuronale Netz frequenzunabhängig. So kann es auch Änderungen in hochfrequenten Spektren erlernen und ist nicht auf einen bestimmten Hörbereich eingeschränkt. Da das Frequenzspektrum für Sprache im Wesentlichen bis 8000 Hz reicht, eignen sich Sprachsignale für diese Untersuchung nicht. Daher wurden zur Schätzung der Distanz Rauschsignale verschiedener Frequenzbänder verwendet.

Mit Erkenntnissen und ähnlichen Netzkonditionen der Vorstudie schließt sich eine umfassende Untersuchung einer sphärischen Schalllokalisation von Rausch- und Sprachsignalen an. Es sollen Richtungen für ein frontales und seitliches Fenster mit jeweils 49 verschiedenen Winkeln des Schalleinfalls detektiert werden. Darüber hinaus wird der Vorteil der zusätzlichen Verwendung vortrainierter Netzwerke zur Extraktion von Bottleneck-Merkmalen evaluiert, sowie verschiedene Arten von Ausgangsvektoren auf ihre Eignung geprüft.

In Anlehnung an Neti et al. [NYS92] wird unter Verwendung von Rauschsignalen vergleichend herausgestellt, ob eine abhängige Betrachtung von Azimuth und Elevation einer unabhängigen Betrachtungsweise je Merkmal vorzuziehen ist. Nimmt man eine Abhängigkeit an, so müssen Netze für jeweiliges Merkmal mit den aufbereiteten Daten des gesamten Kugelausschnittes trainiert werden. Handelt es sich um ein in den Dimensionen unabhängiges Merkmal sollte es genügen dem Netz zur Schätzung der 49 Richtungen nur Informationen der Hauptachsen zum Training bereitzustellen. Letzteres bedeutet eine deutliche Kompression und Vereinfachung im Training.

Extrahierte Merkmale, die dem Netz als Eingang dienen, sind verschiedene Audiorepräsentationen: ILD und ITD einzelner auditorischer Bänder, das binaurale Betrags- und Phasenspektrum, Real- und Imaginärteil des Signalspektrums sowie die Mel-Frequenz-Cepstrum-Koeffizienten. Phasenspektrum, Real-



und Imaginärteil wurden nur in der Vorstudie betrachtet, während die Mel-Frequenz-Cepstrum-Koeffizienten erst in der fortgeschrittenen Studie als geeignete Audiorepräsentation mit einbezogen wurden.

## Kapitel 2

# Stand der Forschung

### 2.1 Schalllokalisation

Lokalisation meint das Zuordnen einer Schalleinfallrichtung und einer Entfernung zu einer primären Schallquelle und ist zunächst Funktion der auditiven Wahrnehmung von Menschen bzw. Lebewesen mit komplexem Gehör. Der Begriff kann ausgeweitet werden auf die technische Lokalisation durch Computer oder Maschinen. In der Literatur spricht man häufig von Lokalisation und meint lediglich das Bestimmen der Schalleinfallrichtung, die sich im sphärischen Koordinatensystem in Azimuth  $\vartheta$  und Elevation  $\varphi$  ausdrücken lässt. Viele Studien beziehen sich auch nur auf das horizontale Lokalisieren des Azimuthwinkels oder aber der vertikalen Lokalisation der Elevation.

### 2.2 Kopfbezogene Übertragungsfunktionen

Kopfbezogene Impulsantworten (Head Related Impulse Responses HRIRs) sind binaurale Signalpaare, die den akustischen Einfluss - Reflexionen- und Abschattungseffekte, Laufzeitunterschiede - der Kopf- und Pinnageometrie enthalten. Ein ausgesendetes Signal wird, ehe es am Trommelfell des Hörenden ankommt, mit diesen Impulsantworten gefaltet. Die Richtungsabhängigkeit von HRIRs ermöglicht die Schalllokalisation. Kopfbezogene Übertragungsfunktionen (Head Related Transfer Functions HRTFs) sind das spektrale Äquivalent der kopfbezogenen Impulsantworten im Zeitbereich. Gemessen werden die HRTFs am Kunstkopf entweder vor linken und rechten verschlossenen Gehörgang, wobei die Gehörgänge akustisch vernachlässigt werden, oder aber an den Trommelfellen. Am echten Menschen gemessen, ist der Einfluss des Torsos inbegriffen wie auch bei Kopf-Torso-Simulatoren. Jeder Mensch verfügt über individuelle HRTFs, da Ohrmuscheln, Kopf und Torso individuell geformt sind (siehe Nicol 2010 [Nic10, vgl. S. 11]). Das menschliche Gehirn wertet auditiv zeitliche und spektrale Eigenschaften dieses binauralen Filters aus, um räumliche Informationen zu erhalten (siehe Blauert 2008 [BB08]).

Eine HRTF einer bestimmten Position kann mittels Anregesignal (Sweep oder Impuls) im reflexionsarmen Raum gemessen werden. Dazu wird der Lautsprecher auf etwaige Position gebracht und Kopf/Torso relativ dazu orientiert. Das Anregesignal wird ausgesendet und aufgenommen an Trommelfellen bzw. Ohrkanälen. Das

Referenzsignal ist das am interauralen Zentrum aufgenommene Anregesignal. Um reine HRIRs bzw. HRTFs zu erhalten, werden zuvor gemessene Signale mit dem Referenzsignal zur Entfernung des Anregesignals und weiterer Störeffekte entfaltet. Bei Møller (1992 [Mø92]) werden Einflussgrößen räumlichen Hörens beschrieben: die monaurale spektrale Färbung der HRTF, die interauralen Phasen- (IPD) und Zeitunterschiede (ITD), sowie Pegeldifferenzen (ILD).

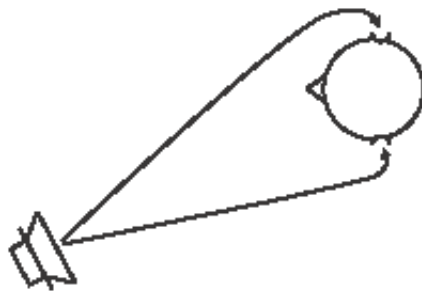


Abbildung 2.1: Skizze des Schallwegs zum Trommelfell

Der Weg eines Signals vom Lautsprecher zu den Trommelfellen (Abb. 2.1) kann unterschiedlich lang sein, wodurch sich Zeit- und Phasenunterschiede ergeben. Phasenunterschiede sind in den tiefen Frequenzen wirksam, Zeitunterschiede in den mittleren Frequenzen. Im hohen Frequenzbereich ergibt sich durch die Abschattung des Kopfes eine Dämpfung. Die spektrale Färbung der HRTF ergibt sich aus Reflexionen, Resonanzen, Streuungen und Abschattungen an den Körperstrukturen.

## 2.3 Abhängigkeit von Azimuth und Elevation

Für die Klassifikation von Richtungsinformationen können verschiedene Merkmale herangezogen werden. Für ein besonders effizientes Training des neuronalen Netzes sind dimensionsunabhängige Merkmale wünschenswert. Es lohnt sich demnach die Abhängigkeit von Azimuth- und Elevationsinformationen zu untersuchen.

Neti et al. [NYS92] legten dazu bereits 1992 eine Studie mit HRTFs von Katzen vor. Wenn Elevation und Azimuth unabhängig für ein neuronales Netzwerk codiert sind, so gingen die Autoren aus, dann sollte das Modell in der Lage sein, von den beiden Eingangsmustern der Hauptachsen auf die jeweilige Richtung der Kugeloberfläche wie in Abbildung 3.1 dargestellt schließen zu können. Demnach würde das System das Muster für die Elevation aus den Informationen der vertikalen Ebene erlernen und das Muster für den Azimuth aus dem bereitgestellten Trainingsset der Horizontalen.

Es konnte gezeigt werden, dass sich untersuchte Merkmale mit beiden Dimensionen Azimuth und Elevation verändern in einer voneinander abhängigen Art. Bei fester

Elevation also kann sich ein bestimmtes Merkmal mit dem Azimuth ändern. Andererseits, so wurde festgestellt, gibt es ebenso Merkmale, die sich bei nur geringer Änderung über die vertikale Auslenkung unabhängig davon in der Horizontalen abwandeln. Beim Richtungslokalisieren einer Schallquelle bei gegebenem Merkmal ist demnach entweder eine mögliche Abhängigkeit der Dimensionen zu berücksichtigen oder aber es kann die Unabhängigkeit als Vorteil genutzt werden.

## 2.4 Neuronale Netze

Einführende Literatur findet sich zum Beispiel von Akira Hirose 2006 [Hir06], Stoica-Klüver et al. 2009 [SKKS09] und Kruse et al. 2012 [KBK<sup>+</sup>12].

Ein neuronales Netz besteht aus vielen Neuronen, die in mehreren Ebenen angeordnet sind. Ein vorwärts gerichtetes Netz verbindet jedes Neuron einer Ebene mit jedem Neuron der nächst folgenden Ebene stets nur in eine Richtung. Ein Neuron stellt dabei eine parametrisierte beschränkte Funktion  $f(\theta_1^{in}, \theta_2^{in}, \dots, \theta_A^{in}, w_1, w_2, \dots, w_A)$  dar, die von Eingaben  $\theta_a^{in}$  und Gewichten  $w_a$  mit  $a = 1, 2, \dots, A$  abhängt.

Bei einem vorwärts gerichteten Durchlaufen des Netzes, wie es in Abbildung 2.2 dargestellt ist, wird das Set der Eingangswerte jeder Ebene  $\theta^{in}$  gewichtet und aufsummiert (Gl. (2.2)). Im Folgenden wird das Ergebnis dessen  $\theta^{net}$  weitergegeben in die Aktivierungsfunktion. Hier ist eine logistische Sigmoidfunktion (Gl. (2.1)) als Aktivierungsfunktion gewählt worden, die anders als eine Stufenfunktion an jeder Stelle differenzierbar ist für das noch vorgestellte Lernverfahren der Fehlerrückführung (Backpropagation 2.4.2).

Abbildung 2.2 zeigt eine schematische Darstellung eines mehrschichtigen neuronalen Netzes. Die Parameter  $N$ ,  $K$  und  $M$  stehen für die Anzahl an Neuronen der entsprechenden Ebene. Die versteckten Ebenen zwischen Eingangs- und Ausgangsschicht enthalten in dieser Darstellung entsprechend jeweils gleich viele Neuronen. Der Vektor  $\mathbf{w}$  definiert die Gewichte der Verbindungen zwischen Neuronen benachbarter Ebenen. Die Variable  $b$  steht für das Biaselement einer Ebene - ein ausgleichendes Gewicht. Alle Ausgangswerte  $\theta^{out}$  der vorangegangenen Ebene sind entweder die Eingangswerte der folgenden Ebene oder aber die Ausgangswerte des Netzes.

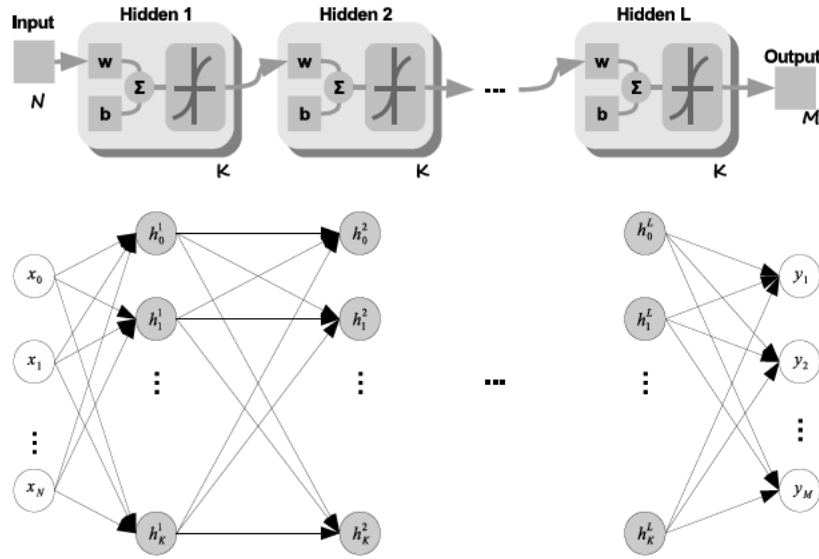


Abbildung 2.2: Schematische Darstellung eines mehrschichtigen neuronalen Netzes mit L versteckten Ebenen.

Der Ausgang  $\Theta_j^{out}$  eines jeden Neurons j ist definiert, wie in Gleichung 2.1 gezeigt. In eine Aktivierungsfunktion  $\phi$  wird die Funktion  $\Theta^{net}$  (Gl. 2.2) gegeben.

$$\Theta^{out} = \phi(\Theta^{net}) = \frac{1}{1 + \exp(-\Theta^{net})} \quad (2.1)$$

$$\Theta^{net} = \sum_{a=1}^A (w_a \cdot \Theta_a^{in}) + b \quad (2.2)$$

$A = n$  falls Ausgangswerte der ersten versteckten Ebene berechnet werden, anderenfalls  $A = m$ .

Das Bias-Neuron ist ein zusätzliches Neuron und erhält daher den Index 0. Es kann in ein Gewicht verwandelt werden. Wodurch sich  $\Theta^{net}$  wie in Gleichung 2.4 darstellt.

$$b = (w_0 \cdot \Theta_0^{in}) \quad (2.3)$$

$$\text{mit } \Theta_0^{in} = 1$$

$$\Theta^{net} = \sum_{a=0}^A (w_a \cdot \Theta_a^{in}) = \vec{w} \cdot \vec{\Theta}^{in} \quad (2.4)$$

$$\text{mit } \vec{w} = (w_0, w_1, \dots, w_A)$$

$$\text{und } \vec{\Theta}^{in} = (\Theta_0^{in} = 1, \Theta_1^{in}, \dots, \Theta_A^{in})$$

### 2.4.1 Bottleneck

Abbildung 2.3 zeigt ein mehrschichtiges Netzwerk mit Bottleneck. Eine innere, versteckte „Bottleneck-Ebene“ eines neuronalen Netzes ist eine Ebene mit reduzierter Neuronenanzahl  $F$ . Umliegende Ebenen enthalten mehr Neuronen. So beschreibt es Kramer bereits 1991 [Kra91]. Er setzt die „Bottleneck-Ebene“ in die Mitte eines fünf-schichtigen Netzes. Diese Einschnürung, die an einen Flaschenhals erinnert, stellt eine Dimensionsreduktion der Daten dar. Das neuronale Netz erhält demnach die Fähigkeit zur nichtlinearen Kompression der Eingangsmerkmale und zur Klassifikation dieser komprimierten Merkmale wie Grez et al. 2007, 2008 beschreiben [GKKC07] [GF08]. In letzterer Studie zu Spracherkennung wurden Netzwerke mit fünf Schichten verwendet, die ebenfalls in ihrer Mitte die Bottleneck-Ebene enthielten. Bottleneck-Merkmale erzielten bessere Ergebnisse als Merkmale, die mit probabilistischen Methoden gewonnen wurden in allen Belangen. Die Einschnürung ist, wie in dieser Arbeit verwendet, auch am Ende als letzte versteckte Ebene des Netzes denkbar. Die Ausgangswerte der Bottleneck-Ebene stellen die Bottleneck-Features dar. Oft werden diese berechnet, um mit den reduzierten Daten weiterzufahren. Es können sich statistische Methoden anschließen oder aber ein weiteres neuronales Netz wie in dieser Arbeit.

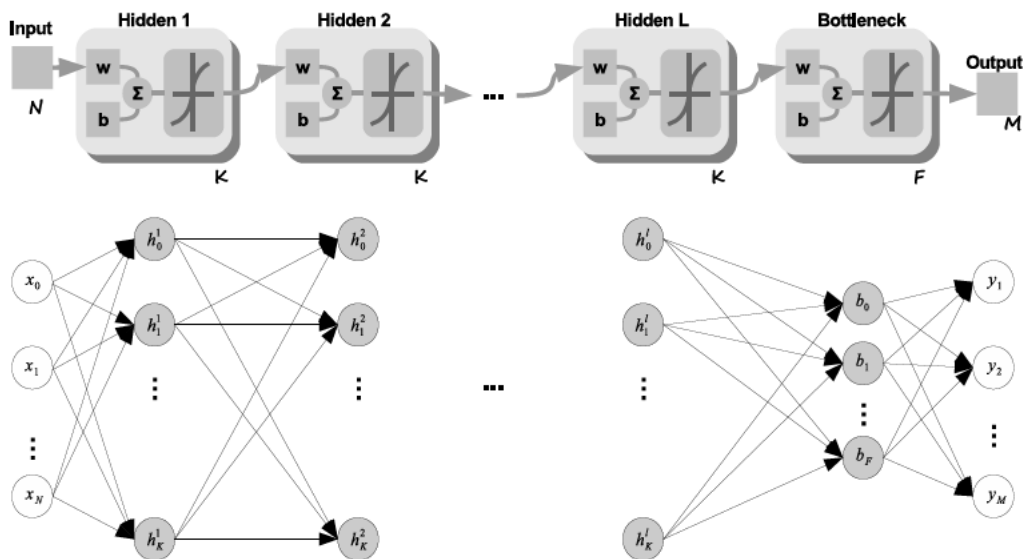


Abbildung 2.3: Schematische Darstellung eines mehrschichtigen neuronalen Netzes mit Bottleneck der letzten Ebene vor dem Ausgang.

### 2.4.2 Backpropagation als Lernverfahren

Muster, die vom neuronalen Netz erkannt werden, sind zunächst trainiert worden, sodass zu einem Eingang der möglichst passende Ausgang zugeordnet wird. Diese Zuordnung geschieht über die Gewichte, die unter anderem mittels

Backpropagation-Algorithmus angepasst werden können über das Verfahren des Gradientenabstiegs. Ziel dabei ist es das Minimum der Fehlerfunktion zu finden. Wenn die Richtung bekannt ist, in die der Fehler kleiner wird, so bewegt man sich ein kleines Stück in diese Richtung, passt die notwendigen Änderungen an, bestimmt erneut die Richtung an neuer Position, solange bis das Minimum der Fehlerfunktion gefunden ist bzw. für eine bestimmte Anzahl von Iterationen. Voraussetzung für dieses Verfahren ist eine differenzierbare Aktivierungsfunktion wie etwa die logistische Funktion. Daraus ergibt sich eine ebenso differenzierbare Fehlerfunktion unter der Bedingung, dass auch die Ausgabefunktion differenzierbar ist. In dieser Arbeit handelt es sich bei der Ausgabe- um die Aktivierungsfunktion. Über den Gradienten der Fehlerfunktion kann daher die Richtung bestimmt werden, in die die Gewichte angepasst werden müssen.

Ein Training bzw. ein Pseudocode des Netzwerkes enthält folglich diese Schritte, die für eine bestimmte Anzahl an Iterationen wiederholt werden oder durch ein Abbruchkriterium enden:

1. Die Gewichte werden mit Werten zwischen  $w = \pm 0.1$  zufällig initialisiert
2. Ein Eingangsvektor der Datenbank wird durch das Netz gegeben, wie es in Abbildung 2.2 und 2.3 dargestellt ist.
3. Der Ausgang des Netzwerkes wird mit dem Sollwert verglichen.
4. Der mittlere quadratische Fehler wird zurückgeführt zum Eingang des Systems.
5. Durch das Verfahren des Gradientenabstiegs wird der Fehler als Funktion der Netzwerkgewichte kleiner.

Der Gradient (Gl. 2.5) der Fehlerfunktion  $E$  kann als partielle Ableitung durch die Kettenregel aus drei Teilen (Gl. 2.6, 2.7, 2.8) gebildet werden.

$$\frac{\partial E}{\partial w_{ij}} = \underbrace{\frac{\partial E}{\partial \Theta_j^{out}}}_I \underbrace{\frac{\partial \Theta_j^{out}}{\partial \Theta_j^{net}}}_{II} \underbrace{\frac{\partial \Theta_j^{net}}{\partial w_{ij}}}_{III} \quad (2.5)$$

$$I) \frac{\partial E}{\partial \Theta_j^{out}} = \frac{\partial}{\partial \Theta_j^{out}} \frac{1}{2} (\Theta_{j,target}^{out} - \Theta_j^{out})^2 = \Theta_j^{out} - \Theta_{j,target}^{out} \quad (2.6)$$

$$II) \frac{\partial \Theta_j^{out}}{\partial \Theta_j^{net}} = \frac{\partial \phi(\Theta_j^{net})}{\partial \Theta_j^{net}} = \phi(\Theta_j^{net})(1 - \phi(\Theta_j^{net})) = \Theta_j^{out}(1 - \Theta_j^{out}) \quad (2.7)$$

$$III) \frac{\partial \Theta_j^{net}}{\partial w_{ij}} = \Theta_j^{in} = \Theta_i^{out} \quad (2.8)$$

Der Ausgang der vorangegangenen Ebene ist der Eingang der Folgenden ( $\Theta_i^{out} = \Theta_j^{in}$ ). Handelt es sich bei der  $i$ -ten Ebene um die erste Ebene, so ist  $\Theta_j^{in}$  ein Vektor des aufbereiteten Datensatzes und Eingang des Netzes.

Handelt es sich bei der  $j$ -ten Ebene um die Ausgangsschicht des neuronalen Netzes gilt für die Änderung der Gewichte  $\Delta w_{ij}$  - wie in Gleichung 2.9 notiert, das Produkt aus negativem Gradient und einem Faktor  $\eta$ , der die Lernrate angibt. Der Gradient

wird negativ, da er in die Richtung des größten Fehlers zeigt und die entgegengesetzte Richtung zur Fehlerminimierung notwendig ist. Die Lernrate  $\eta$  reguliert die Größe der vorzunehmenden Anpassung von Iteration zu Iteration.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta (\Theta_j^{out} - \Theta_{j,target}^{out}) \Theta_j^{out} (1 - \Theta_j^{out}) \Theta_j^{in} \quad (2.9)$$

Liegen die Neuronen der  $j$ -ten Ebene in einer versteckten Schicht, ist die Änderung dieser Gewichte abhängig von der nachfolgenden Ebene und deren Fehler. Verallgemeinernd kann die Gewichtsänderung in Gleichung 2.10 und 2.11 notiert werden. Der Index  $k$  gehört zu den direkt nachfolgenden Neuronen des  $j$ -ten Neurons für den Fall, dass das  $j$ -te Neuron einer inneren versteckten Schicht angehört.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \Theta_i^{out} \delta_j \quad (2.10)$$

$$\delta_j = \begin{cases} (\Theta_j^{out} - \Theta_{j,target}^{out}) \Theta_j^{out} (1 - \Theta_j^{out}) & \text{falls } j \text{ ein Neuron der Ausgangsschicht ist} \\ \sum_k \delta_k w_{jk} \Theta_j^{out} (1 - \Theta_j^{out}) & \text{falls } j \text{ ein Neuron einer versteckten Schicht ist} \end{cases} \quad (2.11)$$

Der Algorithmus kann durch verschiedene Verfahren optimiert werden z.B. durch eine variable Lernrate oder einen Trägheitsterm, der die aktuelle Änderung der Gewichte von der Änderung vorheriger Iteration abhängig macht. Darüber hinaus können verschiedene Aktivierungsfunktionen in versteckten Ebenen und ausgebenden Schicht zu besseren Ergebnissen führen.



# Kapitel 3

## Datensatz

Für die Schalllokalisation von Rausch- und Sprachsignalen werden mit HRTFs gefaltete Sprach- und Rauschsignale einer Merkmalsextraktion unterzogen, deren Ergebnis ein Datensatz von Eingangsvektoren zum neuronalen Netz sind. Im Folgenden werden einzelne Teile und Schritte zur Erstellung dieses Datensatzes beschrieben. Der Quellcode ist dazu im Anhang C i zu finden.

### 3.1 HRTF

Wie im vorangegangenen Kapitel bereits beschrieben, sind HRTFs fouriertransformierte, paarweise Impulsantworten, die beschreiben wie ein akustisches Quellsignal am linken und rechten Trommelfell bzw. am linken und rechten Ohrkanal durch den Kopf und den Torso in Abhängigkeit der Schalleinfallrichtung verändert wird. HRTF-Datensätze sind ein wichtiges Werkzeug räumliches Hören des Menschen zu erforschen sowie die Entwicklung computerbasierter Algorithmen zur Simulation räumlicher Höreindrücke oder auch Unterstützung des räumlichen Hörens voranzutreiben.

Der in dieser Arbeit verwendete HRTF-Datensatz ist als Teil der Datenbank, die von Thiemann et al. 2015 [TvdP15] erstellt wurde, entnommen. Zur Aufnahme der HRTFs verschiedener Kunstköpfe wurden der Messaufbau und Messroutinen nach Brinkmann et al. 2013 [BLW<sup>+</sup>13] adaptiert. Es werden für eine hohe Auflösung von 2° fast alle möglichen Schalleinfallrichtungen abgedeckt. Lediglich Winkel der Polkappen der Kugeloberfläche konnten nicht gemessen werden und entfallen daher. In dieser Arbeit ist für einen KEMAR-Kunstkopf mit Torso zunächst eine kleinere Auflösung verwendet worden bei zwei verschiedenen Fensterausschnitten der Kugeloberfläche. So wird einmal ein frontales Fenster zwischen einem Azimuth von -30 bis +30° und zwischen einem Elevationswinkel von -10 bis +50° in jeweils 10°-Schritten betrachtet. Ein weiterer Kugelausschnitt liegt seitlich für den Azimuth von 60 bis 120° bei gleicher Elevation und gleichem Abstand zwischen den Schalleinfallswinkeln. Je Fensterausschnitt werden demnach 49 Punkte auf der Kugeloberfläche betrachtet. Diese Fenster wurden gewählt, um zum einen den frontalen Standardfall abzudecken, wobei der Mensch besonders empfindlich für Veränderungen der frontalen Schalleinfallrichtungen ist. Daraus kann vermutet werden, dass zugehörige HRTFs ein hohes Maß an Informationen aufweisen, die wiederum zu bestmöglichen Ergebnissen bei der Lokalisation durch ein neuronales

Netzwerk führen sollten. Zum anderen ist der seitliche Fall ausgewählt, wo es unter Umständen zu Vorn-Hinten-Vertauschungen kommen kann. Der sogenannte „Cone of Confusion“ ist bei Algazi et. al [ADMT01] beispielsweise gezeigt. Damit ist ein eher konservativer Fall für die Lokalisation mittels neuronalen Netzwerk abdeckt. Verwendete HRTFs sind in Abbildung 3.1 gezeigt.<sup>1</sup>

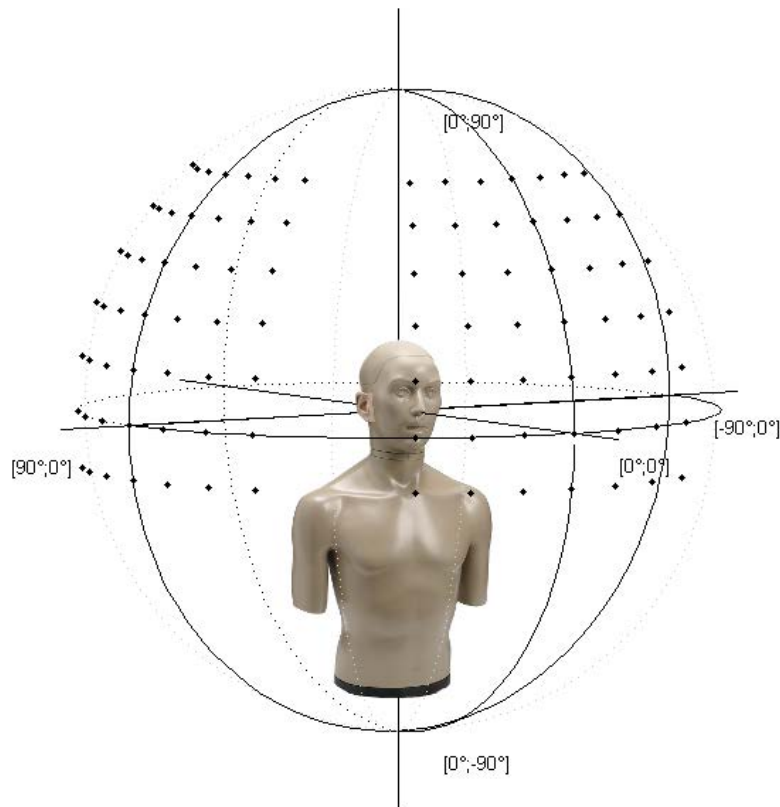



Abbildung 3.1: verwendete HRTFs

Das  **HRTF\_Processing** (vgl. Anhang C i) beinhaltet maßgeblich die Entnahme der Daten gewünschter Richtungen aus dem original HRTF-Datensatz, das Kürzen der HRIRs auf 425 Samples und das Fenstern mit Sinus- bzw. Cosinusflanke. Original-Dateien sowie die hier verwendete gekürzte Version des Subdatensatzes liegen auf dem Datenträger anbei D vii.

## 3.2 Rauschsignale


Es sind mehrere verschiedene Rauschsignale von einer Sekunde Länge generiert worden. Es handelt sich um rosa Rauschen, dessen Leistungsspektrum über die

<sup>1</sup>KEMAR-Abbildung <http://www.aimil.com/Resources/Products/Original/1290.jpg>


Frequenz  $f$  mit  $1/f$  abnimmt. Der Quellcode ist im Anhang C iii ersichtlich. Verwendete Rausch-Signale liegen der Vollständigkeit halber ebenfalls auf dem Datenträger anbei D vii.

### 3.3 Sprachsignale

Um die neuronalen Netze hinsichtlich ihrer Schalllokalisationsleistung für Sprache zu testen, sind Logatome des Oldenburger Logatomkorpuses entnommen worden. Logatome sind kurze Wörter bestehend aus drei Phonemen. Die Datenbank wird bei Wesker et al. 2005 ([WMW<sup>+</sup>05]) genau vorgestellt. Der Korpus enthält neben den Sounddateien eine detaillierte Beschreibung, Wortlisten, Label-Dateien, technische Spezifikationen, sowie Daten zur Kalibrierung. Für einzelne Sprecher mit jeweiligen Dialekten bzw. dialektfreier Sprache sind die dazugehörenden Unterordner zu finden. Es handelt sich um ca. 140000 Dateien, die einer ungefähren Gesamtspieldauer von 60 Stunden entsprechen. Die Datenbank ist unter <http://medi.uni-oldenburg.de/ollo> (zuletzt gesehen am 15.07.2015) zugänglich. Die Datenbank wird in der Literatur für automatische Sprachverarbeitung oder auch für Spracherkennung verwendet ([MZR08] [MJW<sup>+</sup>10] [MBK11]).

Für diese Arbeit wurden mit weiblicher dialektfreier Stimme eingesprochene Logatome verwendet. Die neuronalen Netze sind entsprechend für dieses Sprachmaterial trainiert und getestet worden. Die Samplingrate beträgt 16 kHz. Stille Anteile sind aus den Signalen für diese Arbeit entfernt worden ( `prep_sphere_dataset_function`), um ein aufwendiges Labeln einzelner Sequenzen ersparen zu können. Die verwendeten Logatome sind wie die Rauschsignale auf dem Datenträger vii zu finden.

### 3.4 Binaurale Signale

Um Richtungsinformationen enthaltende binaurale Sprach- bzw. Rauschsignale zu erhalten, werden zunächst die HRTF-Daten auf die gleiche Samplingfrequenz von 16 kHz gebracht. Anschließend werden rechter und linker Kanal der einzelnen HRTF-Signalpaare mit den vorliegenden einkanaligen, monauralen Sprach- und Rauschdateien gefaltet ( `prep_sphere_dataset_function`). Damit sind Betrachtungen des Frequenzspektrums bis 8 kHz möglich.

### 3.5 Ausbreitungsdämpfung zur Simulation von Entfernungen

Da keine HRTFs für verschiedene Entfernungen vorliegen, wird der Versuch unternommen mittels einer errechneten Ausbreitungsdämpfung verschiedene Entfernungen zu simulieren. Für fünf Distanzen ( $r \in \{10, 20 \dots 50\}$  m) ist die atmosphärische Ausbreitungsdämpfung bzw. sind die frequenzspezifischen Koeffizienten [dB/m] auf Grundlage der ISO 9613-1 berechnet worden (Formeln 3.1-3.3 vgl. ISO 9613-2 [ISO99] Formeln (3-5) pure tone attenuation coefficient). Einbezogen werden dabei die Relaxationsfrequenzen von Sauerstoff  $f_{rO}$  und Stickstoff  $f_{rN}$ , sowie der vorherrschende Luftdruck  $p_a$  in Bezug zu einem

Referenzluftdruck mit  $p_r = 101.325 \text{ kPa}$ ; wobei unter Normalbedingungen für  $p_a = p_r$  angenommen wurde. Des Weiteren wird die relative Luftfeuchtigkeit  $h$  mit einem typischen Wert von  $h=50 \%$  und die Temperatur  $T$  unter Normalbedingungen mit  $T=T_0$  zur Referenz von  $T_0 = 293.15 \text{ K}$  gesetzt. Die Dämpfungskoeffizienten  $\alpha$  ergeben sich abhängig von der Frequenz  $f$ .

$$f_{rO} = \frac{p_a}{p_r} (24 + 4.04 \cdot 10^4 h \frac{0.02 + h}{0.391 + h}) \quad (3.1)$$

$$f_{rN} = \frac{p_a}{p_r} \left( \frac{T}{T_0} \right)^{-\frac{1}{2}} (9 + 280 h^{-4.170[(\frac{T}{T_0})^{-\frac{1}{3}} - 1]}) \quad (3.2)$$

$$\begin{aligned} \alpha = & 8.686 f^2 \left( [1.84 \cdot 10^{-11} \left[ \frac{p_a}{p_r} \right]^{-1} \left[ \frac{T}{T_0} \right]^{\frac{1}{2}}] + \right. \\ & + \left[ \frac{T}{T_0} \right]^{-\frac{5}{2}} (0.01275^{-\frac{2239.1}{T}} \left[ f_{rO} + \frac{f^2}{f_{rO}} \right]^{-1} + \\ & \left. + 0.01068^{-\frac{3352.0}{T}} \left[ f_{rN} + \frac{f^2}{f_{rN}} \right]^{-1} \right) \end{aligned} \quad (3.3)$$

Die errechneten Werte sind dem Spektrum der mit Rauschsignalen gefalteten HRTF frontaler Richtung mit Azimuth  $\vartheta = 0^\circ$  und Elevation  $\varphi = 0^\circ$  zugefügt worden. Die Ausbreitungsdämpfung nimmt mit der Frequenz zu und ist im Bereich bis 8 kHz noch sehr gering. In der Vorstudie soll überprüft werden, ob ein neuronales Netz diese empfindlichen Unterschiede lernen kann. Diese Untersuchung beschränkt sich zunächst auf Rauschsignale, die anders als Sprache auch bei höherer Abtastrate höhere Frequenzanteile enthalten. Hier ist ein rosa Rauschen bei einer Abtastrate von  $f_s = 44.1 \text{ kHz}$  gewählt worden. Der Quellcode ist im Anhang C ii ersichtlich.

## 3.6 Merkmalsextraktion

Im Folgenden wird die Merkmalsextraktion aus den binauralen Zeitsignalen  $y_{\{l,r\}}[n]$  (Gl. (3.4)) zu Merkmalsvektoren beschrieben. Der linke und rechte Kanal ist durch das tiefgestellte  $l$  und  $r$  gekennzeichnet; der Zeitindex durch die Variable  $n$ .

$$y_{\{l,r\}}[n] = [y_{\{l,r\}}[n], y_{\{l,r\}}[n-1], \dots, y_{\{l,r\}}[n-N+1]]^T \quad (3.4)$$

Aus dem Zeitsignal extrahierte Merkmalsvektoren enthalten noch die räumlichen Informationen, stellen aber nur noch eine komprimierte Repräsentation der Audiosignale dar. Der selbstgeschriebene Quellcode zur Merkmalsextraktion befindet sich im Anhang C iv - weitere Funktionen und Skripte neben jenen auf dem Datenträger D vii.

### 3.6.1 ILD und ITD

Es werden die interaurale Pegeldifferenz ILD sowie interaurale Zeitdifferenz ITD extrahiert. ILD und ITD werden allgemein nach Gleichung (3.5) und (3.6) berechnet.  $f_s$  beschreibt dabei die Abtastfrequenz.

$$\text{ILD} = 20 \log_{10} \left( \frac{\sqrt{\frac{1}{N} \sum_{i=1}^N y_l^2[i]}}{\sqrt{\frac{1}{N} \sum_{i=1}^N y_r^2[i]}} \right) \quad [\text{dB}] \quad (3.5)$$

$$\text{ITD} = \max \left( \underbrace{\sum_{m=-\infty}^{\infty} y_l^*[m] y_r[n+m]}_{\text{cross correlation}} \right) / f_s \cdot 10^6 \quad [\mu\text{s}] \quad (3.6)$$

Genauer wird der Extraktion noch ein Gammatonefilter nach Slaney 1993 [Sla93], der im Frequenzbereich von 0,1 bis 8 kHz eine Filterbank mit 30 ERB-Filterbändern (equivalent rectangular bandwidth ERB) bereitstellt, vorangeschaltet, sodass ILD und ITD 30 mal berechnet werden. Ein Merkmalsvektor für die ILD bzw. ITD hat daher eine Länge von 30. Werden die Merkmale kombiniert sind es 60, da die Vektoren einfach hintereinander geschaltet werden.

### 3.6.2 Spektrale Merkmale

#### Betragsspektrum

Weitere Audiorepräsentation ist das binaurale Betragsspektrum  $|\mathcal{F}(y_{\{l,r\}})|$ , das mittels Kurzzeit-Fourier-Transformation (short-term fourier transformation STFT) des Zeitsignals berechnet wurde. Bei einer STFT-Länge von 512 Samples und einer Abtastrate von 16 kHz, ergeben sich 257 Bins je Kanal des Signals. Für rechten und linken Kanal zusammen erhält der Merkmalsvektor eine Länge von 514.

#### Phase, Real- und Imaginärteil

Die Phase  $\arg(\mathcal{F}(y_{\{l,r\}}))$ , der Real-  $\text{Re}(\mathcal{F}(y_{\{l,r\}}))$  und Imaginärteil  $\text{Im}(\mathcal{F}(y_{\{l,r\}}))$  wurde unter Verwendung der STFT des Zeitsignals berechnet. Wird eine STFT-Länge von 512 Samples bei einer Abtastfrequenz von  $f_s = 16$  kHz angenommen, enthält das halbe Spektrum 257 Bins für den rechten oder linken Kanal des Signals. Für eine Kombination des Betrags und der Phase zu einem DNN-Eingangsvektor, entspricht demnach die Anzahl an Eingangsneuronen vier mal der Binanzahl des halben Spektrums, da zwei Merkmale für beide Kanäle hintereinander zu einem Vektor kombiniert werden.

### 3.6.3 MFCC

Das Signal wird in sich überlappende zeitliche Fenster zerlegt und durch die Fouriertransformation je Fenster in ein Betragsspektrum ohne Phase überführt. Nach Logarithmieren der Betragsspektren und anschließender Reduktion der Frequenzbänder als Abbildung auf die Mel-Skala können durch eine Dekorrelation per Kosinustransformation die Mel-Frequenz-Cepstrum-Koeffizienten (Mel Frequency Cepstral Coefficients MFCC) gewonnen werden. Matlabskript zur Berechnung der Koeffizienten ist der frei zugänglichen Toolbox „Voicebox“ entnommen.

### 3.7 Targetvektoren

Alle vorangegangene Abschnitte in diesem Kapitel zu HRTFs, Rausch- und Sprachsignalen, sowie zur Merkmalsextraktion beziehen sich alle auf das Erstellen der Eingangsvektoren für das neuronale Netz. Es soll kurz die Codierung der Targetvektoren geschildert werden. Diese ordnet Neuronen der Targetvektoren bestimmte zu erkennende Punkte im Koordinatensystem zu. Das neuronale Netz soll während des Trainings jeden einzelnen Ausgangsvektor möglichst genau den jeweiligen Ziel- bzw. Targetvektor angleichen. Das tut es wie in Abschnitt 2.4 beschrieben über die Veränderung der Gewichte mittels Fehlerrückführung.

In dieser Arbeit werden drei verschiedene Codierungen verwendet, die zu verschiedenen Vektorlängen führen: den parallelen Einzelfall von Azimuth und Elevation, die aneinandergehängte Kombination aus Azimuth und Elevation und eine sphärische Codierung aller Punkte eines Fensterausschnittes innerhalb eines Vektors. Die Variable  $n$  sei hier die Neuronennummer. Winkelangaben für Elevation  $\varphi$  und Azimuth  $\vartheta$  sind angegeben in  $^\circ$ . Für zwei verwendete Fensterausschnitte ergeben sich mehrere Vektoren.

- Azimuth einzeln  $n[\vartheta]$

1[-30] 2[-20] 3[-10] 4[0] 5[10] 6[20] 7[30]

1[60] 2[70] 3[80] 4[90] 5[100] 6[110] 7[120]

- Elevation einzeln  $n[\varphi]$

1[-10] 2[0] 3[10] 4[20] 5[30] 6[40] 7[50]

- Azimuth und Elevation aneinander gehängt,  
für  $n=1, 2 \dots 7$   $n[\vartheta]$ , für  $n=8, 9 \dots 14$   $n[\varphi]$

1[-30] 2[-20] 3[-10] 4[0] 5[10] 6[20] 7[30] 8[-10] 9[0] 10[10] 11[20] 12[30]  
13[40] 14[50]

1[60] 2[70] 3[80] 4[90] 5[100] 6[110] 7[120] 8[-10] 9[0] 10[10] 11[20] 12[30]  
13[40] 14[50]


- sphärisch  $n[\vartheta, \varphi]$

1[-30,-10] 2[-30,0] 3[-30,10] 4[-30,20] 5[-30,30] 6[-30,40] 7[-30,50]  
8[-20,-10] 9[-20,0] 10[-20,10] 11[-20,20] 12[-20,30] 13[-20,40] 14[-20,50]  
15[-10,-10] 16[-10,0] 17[-10,10] 18[-10,20] 19[-10,30] 20[-10,40] 21[-10,50]  
22[0,-10] 23[0,0] 24[0,10] 25[0,20] 26[0,30] 27[0,40] 28[0,50]  
29[10,-10] 30[10,0] 31[10,10] 32[10,20] 33[10,30] 34[10,40] 35[10,50]  
36[20,-10] 37[20,0] 38[20,10] 39[20,20] 40[20,30] 41[20,40] 42[20,50]  
43[30,-10] 44[30,0] 45[30,10] 46[30,20] 47[30,30] 48[30,40] 49[30,50]

```

1[60,-10] 2[60,0] 3[60,10] 4[60,20] 5[60,30] 6[60,40] 7[60,50]
8[70,-10] 9[70,0] 10[70,10] 11[70,20] 12[70,30] 13[70,40] 14[70,50]
15[80,-10] 16[80,0] 17[80,10] 18[80,20] 19[80,30] 20[80,40] 21[80,50]
22[90,-10] 23[90,0] 24[90,10] 25[90,20] 26[90,30] 27[90,40] 28[90,50]
29[100,-10] 30[100,0] 31[100,10] 32[100,20] 33[100,30] 34[100,40] 35[100,50]
36[110,-10] 37[110,0] 38[110,10] 39[110,20] 40[110,30] 41[110,40] 42[110,50]
43[120,-10] 44[120,0] 45[120,10] 46[120,20] 47[120,30] 48[120,40] 49[120,50]

```

Die Codierung wird vorgenommen innerhalb der Funktion  `prep_sphere_dataset_function` im Anhang C iv. Beispielsweise würde der Azimuth von  $\vartheta = 10^\circ$  für den parallelen Fall als Targetvektor ausgegeben so erscheinen: `[0 0 0 0 1 0 0]`. Der Punkt  $[\vartheta = -20^\circ, \varphi = 30^\circ]$  ergäbe für die aneinandergehängte Version von Azimuth und Elevation diesen Vektor: `[0 1 0 0 0 0 0 0 0 0 1 0 0]`.

## 3.8 Verwendete neuronale Netze


### 3.8.1 OXlearn

OXlearn ist eine frei zugängliche Software zur Simulation neuronaler Netze. Mittels dieser ist es möglich sehr einfache Modelle konnektionistischer neuronaler Netzwerke zu trainieren, zu testen und zu analysieren. Sie ist als Matlab Toolbox erhältlich. Die Version OXlearn1.1 liegt als Original auf Datenträger dieser Arbeit anbei D vii. Die OXlearn-Toolbox ist vor allem ein Lehrmittel, das einen einfachen und schnellen Start im Modellieren von Neuronalen Netzen ermöglicht. Durch eine graphische Oberfläche kann auf Parameter des Neuronalen Netzes und damit weitestgehend auf dessen Funktionalität zugegriffen werden, ohne Quellcode editieren zu müssen. Im Handbuch, das als PDF in der Toolbox enthalten ist, werden Struktur und Funktionalität, sowie einzelne Parameter beschrieben. Es ist ausdrücklich erlaubt ebenfalls direkt auf den Quellcode Einfluss zu nehmen, um diesen für den eigenen Bedarf anzupassen, wie es in der Einleitung des Nutzer-Handbuches steht. Versionen der OXlearn-Toolbox kann frei für den persönlichen oder wissenschaftlichen Gebrauch weiterverbreitet und verändert werden. Copyright verbleibt bei dem Autor Nicolas Ruh [RW09]. Für diese Arbeit dienten vorhandene Matlab-Dateien der Version OXlearn 1.1 (<http://psych.brookes.ac.uk/oxlearn/>) - kompatibel mit Matlab 2011b oder jünger, als Ausgangspunkt und sind umfassend verändert, erweitert und um andere Files ergänzt worden. Sämtliche Dateien finden sich auf dem Datenträger im Anhang D vii.

### 3.8.2 Änderungen/Erweiterungen in der OXlearn-Toolbox

Selbstgeschriebene und maßgeblich geänderte Skripte finden sich im Anhang C v.

#### Main-Datei

Die geänderte Datei  `Main.m` startet das Training eines etwaigen neuronalen Netzes. Die wesentliche Änderung ist das Unterdrücken des Gui-Aufrufs. Die

Gui diene hauptsächlich der Visualisierung aller Parameter zum einfacheren Erschließen der Funktionsweise des neuronalen Netzes. Im neuen Main-Skript werden Variablen festgelegt, die die Art des Netzes und Parameter des Trainings betreffen, sowie Pfade zu Trainings- und Testdateien und Speicherorte festgelegt. Darunter fallen die Ebenenanzahl, Anzahl der Neuronen in versteckten Schichten und der Bottleneck-Ebene, die kleinste und größte Epochenanzahl<sup>2</sup>, die Lernrate, das Momentum des Trägheitsterms und ein Stopkriterium, das das Training beendet bei ausreichend kleinem Fehler. Es wird in der Main-Datei die Funktion zur Initialisierung aller Gewichte `initWeights.m` im Unterordner „helperfunctions“ aufgerufen. Diese Funktion ist angepasst nach Art des Netzes.

Die Original-Toolbox sieht als Start den Aufruf des Matlabskriptes `OXlearn.m` vor, das lediglich die Original-Main-Datei `OXmain.m` ausführt. Zum Trainieren und Testen der Netzwerke für diese Arbeit genügt das Ausführen der bearbeiteten Main-Datei `MAIN.m` im Editor der Programmiersoftware Matlab der Version 2011a und später.

### Trainingsskripte und Testfunktionen

Je Art des Netzwerkes, ob mit oder ohne Bottleneck, gibt es angepasste Skripte und Funktionen: `OX_trainFFn_wGUI.m`, `OX_trainFFn_bottle_wGUI.m`, `OX_testFFn_wGUI.m`, `OX_testFFn_bottle_wGUI.m`. Sie sind angelehnt an die Toolbox eigenen Dateien `OX_trainFF3.m` und `OX_testFF3.m`. Kern des Netzes ist dabei die jeweilige Trainingsdatei. Sie enthält die Netz- und Aktivierungsfunktion, die Berechnung und Rückpropagierung des Fehlers, sowie die Änderung der Gewichte auf Basis der in Kapitel 2.4 und insbesondere in Abschnitt 2.4 beschriebenen Vorgehensweise. Darüber hinaus wurde ein Trägheitsterm mit Momentum als Optimierung der Fehlerrückführung eingefügt. Das Training erfolgt Ebene für Ebene. Für drei versteckte Ebenen demnach in vier Einzeltrainings. Nach jedem Einzeltraining wird eine weitere Ebene hinzugeschaltet. Handelt es sich um ein Netz mit Bottleneck, so wird zunächst dieses ebenfalls schichtweise trainiert. Ergebnis dessen, die sogenannten Bottleneck-Features werden als Eingang zu einem Netzwerk ohne Bottleneck geführt, das wiederum Ebene für Ebene trainiert wird. Es handelt sich bei der Vorgehensweise mit Bottleneck um ein Vortraining bzw. um eine Dimensionsreduktion der Daten ähnlich einer Hauptkomponentenanalyse, an die sich meist auch eine weitere Untersuchung anschließt.

Das Training kann bei guten Tendenzen frühzeitig durch die Wahl eines Stopkriteriums für den mittleren quadratischen Fehler beendet werden. Dies geschieht frühestens nach dem Durchlaufen einer gewählten Anzahl von Epochen. Wird das Kriterium nicht frühzeitig erreicht, so wird für die festgelegte Anzahl an Epochen weiter trainiert.

---

<sup>2</sup>Eine Epoche entspricht der Anzahl der bereitgestellten Eingangsvektoren für das neuronale Netz



## Kapitel 4

# Vorstudie für einzelne Dimensionen

Zur Ermittlung geeigneter Netzvarianten und Audiorepräsentationen wurde eine Vorstudie mit kleinerer Anzahl an Winkeln für die Horizontal- und Medianebene durchgeführt. Sieben Richtungen wurden jeweils für Azimuth ( $\vartheta \in \{-30^\circ, -20^\circ, \dots, 30^\circ\}$ ) und Elevation ( $\varphi \in \{-10^\circ, 0^\circ, \dots, 50^\circ\}$ ) ausgewählt, wie auch in Abbildung 4.1 dargestellt. Die Simulation von fünf verschiedenen Entfernung ( $r \in \{10, 20 \dots 50\}\text{m}$ ) wurde, wie schon im Unterkapitel 3.5 beschrieben, auf die frontale Einfallsrichtung beschränkt.

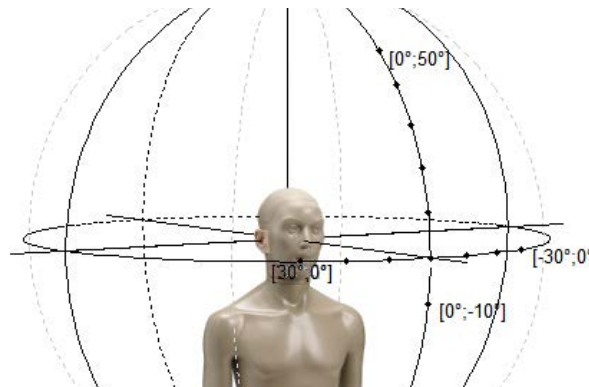


Abbildung 4.1: Grid verwendeter HRIRs,  $[\vartheta; \varphi]$

### 4.1 Untersuchte Merkmale

In Tabelle 4.1 sind die untersuchten Merkmale bzw. deren Kombinationen gelistet. Daraus ergeben sich für das neuronale Netz Eingangsvektoren einer bestimmten Länge, die maßgeblich den Rechenaufwand beeinflussen. Die Längen sind ebenfalls in der Tabelle 4.1 angegeben. Es handelt sich dabei um die einfache ILD und ITD, sowie den kombinierten Fall, der beide Vektoren hintereinander gesetzt zu einem Vektor werden lässt mit entsprechend doppelter Länge. Im Weiteren wird das binaurale Betragsspektrum in einem Vektor, sowie Real- und Imaginärteil in einem

Vektor geführt. Auch die Kombination von Betrag und Phase wird in der Vorstudie dem neuronalen Netz als Eingangsvektor präsentiert.


Da die Distanzen über die Manipulation des Betragsspektrum per Dämpfungskoeffizienten (siehe Absatz 3.5) simuliert wurden, wurde auch das Betragsspektrum als Eingangsvektor für das neuronale Netz gewählt. Es wurde dabei mit verschiedenen Bereichen des Spektrums experimentiert.

Tabelle 4.1: Eingangsvektoren des DNN und deren Länge

| Merkmal   | Länge               |
|---|---------------------|
| ILD   | 30                  |
| ITD   | 30                  |
| ILD, ITD kombiniert   | 30+30               |
| $ \mathcal{F}(y_{\{l,r\}}) $  | 257+257             |
| $\text{Re}(\mathcal{F}(y_{\{l,r\}})), \text{Im}(\mathcal{F}(y_{\{l,r\}}))$ kombiniert | $2 \cdot (257+257)$ |
| $ \mathcal{F}(y_{\{l,r\}}) , \arg(\mathcal{F}(y_{\{l,r\}}))$ kombiniert               | $2 \cdot (257+257)$ |

## 4.2 Netzkonditionen

Für die Evaluation verschiedener Eingangsvektoren wurden die Netzwerkparameter für alle getesteten Fälle konstant gehalten. Zwei versteckte Ebenen wurden jeweils trainiert, wobei die Lernrate  $\eta$  auf den Wert von  $\eta = 0.1$  und für den Trägheitsterm das Momentum auf 0.3 festgelegt wurden. Das Stopkriterium für den zu erreichenden mittleren quadratischen Fehler liegt bei einem Wert von  $10^{-8}$ .

Für jeden untersuchten Fall sind jeweils eine Sammlung von Eingangsvektoren erstellt worden, die aus 100 verschiedenen Logatomen errechnet worden sind. Die Wahl der Logatome aus der Datenbank wurde randomisiert. Ein Anteil von 50% der Eingangsvektoren wurde anschließend jeweils für das Training und das Testen des neuronalen Netzes bereitgestellt. Die Merkmale sind blockweise mit der Blocklänge von 25 ms, einer Überlappung (hop-size) von 10 ms und einer STFT-Länge von 512 Samples bei einer Abtastrate von  $f_s = 16 \text{ kHz}$  berechnet worden (vgl.  `prep_sphere_dataset_function.m`, Anhang C iv).

In der Vorstudie sind die räumlichen Dimensionen unabhängig voneinander für die jeweiligen Inputvektoren betrachtet worden. Wie bereits erwähnt wurden für die horizontale und vertikale Ebene Logatome als Quellsignale verwendet. Zum Training und Testen der Netzwerke für ein Erkennen verschiedener Distanzen diente als Quellsignal ein rosa Rauschen. Nach dessen Falten mit entsprechenden HRTFs lieferten die Betragsspektren zwischen 18 und 22.05 kHz die Eingangsvektoren zum neuronalen Netz. Aufgrund der geänderten Abtastrate zu  $f_s = 44.1 \text{ kHz}$  bei den Untersuchungen zur Entfernung betrug die STFT-Länge 2048 Samples.

Tabelle 4.2 führt alle Bedingungen der Vorstudie auf. Sechs verschiedene Arten von Eingangsvektoren wurden jeweils für horizontale und vertikale Ebene untersucht: jeweils die ILD und ITD in 30 ERB-Filterbändern und die aneinandergehängte Kombination aus ILD- und ITD-Werten, das Betragsspektrum  $|\mathcal{F}(y_{\{l,r\}})|$ , die aneinandergehängte Kombination aus Real- und Imaginärteil  $[\text{Re}(\mathcal{F}(y_{\{l,r\}})), \text{Im}(\mathcal{F}(y_{\{l,r\}}))]$ , sowie die Kombination aus Betrags- und Phasenspektrum  $[|\mathcal{F}(y_{\{l,r\}})|$

$\arg(\mathcal{F}(y_{\{l,r\}}))$ ]]. Ein Fall, der hier gezeigt ist, bezieht sich auf die Distanz (vgl. Tab. 4.2 XIII).

Die Zahl der Iterationen im Training ist festgelegt worden auf  $6 \cdot 10^5$ . Daraus ergibt sich die Epochenanzahl, die ebenfalls der Tabelle entnommen werden kann. Im Weiteren sind die Epochenlänge des Trainings und die Anzahl der Neuronen in Eingangs- und Ausgangsschicht, und in den versteckten Ebenen  $I, H, O$  aufgeführt. Die Epochenlänge variiert mit der randomisierten Wahl der 50 Logatome, die wiederum in ihrer Dauer jeweils variieren und damit in mehr oder weniger Vektoren zerlegt werden.

Tabelle 4.2: In den ersten Spalten finden sich getestete Fälle der Vorstudie mit zugehörigen Parametern. Die letzte Spalte beinhaltet die gemittelte Hit-Rate der gesamten Performance. (Schätzrate beträgt: 14.3% für Horizontal-/Medianebene, 20% für die Entfernung).

|             | Dim                    | Merkmal  | Epochenlänge | Epochenanzahl | I    | H    | O | Ø Hit-Rate [%] |
|-------------|------------------------|--|--------------|---------------|------|------|---|----------------|
| <b>I</b>    | $[\vartheta; 0^\circ]$ | ILD  | 19915        | 30.1          | 30   | 100  | 7 | 99.8           |
| <b>II</b>   | $[\vartheta; 0^\circ]$ | ITD  | 20118        | 29.8          | 30   | 100  | 7 | 99.1           |
| <b>III</b>  | $[\vartheta; 0^\circ]$ | ILD, ITD   | 18959        | 31.6          | 60   | 100  | 7 | 98.8           |
| <b>IV</b>   | $[\vartheta; 0^\circ]$ | $ \mathcal{F}(y_{\{l,r\}}) $   | 19400        | 30.9          | 514  | 600  | 7 | 100            |
| <b>V</b>    | $[\vartheta; 0^\circ]$ | $\text{Re}(\mathcal{F}(y_{\{l,r\}})), \text{Im}(\mathcal{F}(y_{\{l,r\}}))$ | 17633        | 34.0          | 1028 | 1100 | 7 | 100            |
| <b>VI</b>   | $[\vartheta; 0^\circ]$ | $ \mathcal{F}(y_{\{l,r\}}) , \arg(\mathcal{F}(y_{\{l,r\}}))$               | 20209        | 29.7          | 1028 | 1100 | 7 | 14.3           |
| <b>VII</b>  | $[0^\circ; \varphi]$   | ILD  | 19330        | 31.0          | 30   | 100  | 7 | 100            |
| <b>VIII</b> | $[0^\circ; \varphi]$   | ITD  | 19351        | 31.0          | 30   | 100  | 7 | 37.1           |
| <b>IX</b>   | $[0^\circ; \varphi]$   | ILD, ITD   | 19848        | 30.2          | 60   | 100  | 7 | 97.2           |
| <b>X</b>    | $[0^\circ; \varphi]$   | $ \mathcal{F}(y_{\{l,r\}}) $   | 18788        | 31.9          | 514  | 600  | 7 | 94.2           |
| <b>XI</b>   | $[0^\circ; \varphi]$   | $\text{Re}(\mathcal{F}(y_{\{l,r\}})), \text{Im}(\mathcal{F}(y_{\{l,r\}}))$ | 18315        | 32.8          | 1028 | 1100 | 7 | 23.5           |
| <b>XII</b>  | $[0^\circ; \varphi]$   | $ \mathcal{F}(y_{\{l,r\}}) , \arg(\mathcal{F}(y_{\{l,r\}}))$               | 17790        | 33.7          | 1028 | 1100 | 7 | 14.3           |
| <b>XIII</b> | r                      | $ \mathcal{F}(\text{noise}) $  | 12000        | 50            | 190  | 200  | 5 | 71             |

### 4.3 Allgemeine Vorgehensweise der Evaluation

Die Skripte zur Evaluation der trainierten Netze finden sich im Anhang C vi. Das Skript `eval_test_data_2D.m` greift dabei zurück auf die Funktion `eval_mean_frame.m`. Hierbei wird das jeweilige trainierte Netz mit dem für Testzwecke zurückgelegten Teil generierter Netz-Eingangsvektoren von 50% getestet und ausgewertet. Mittlere Hit-Raten fassen das Ergebnis des Testdurchgangs in einem Wert zusammen. Zur Berechnung einer einfachen Hit-Rate wurde das Mittel aus zehn Frames (entsprechend 100 ms) als eine Entscheidung des Netzwerkes gewertet. Dazu wurden immer zehn Netz-Ausgangsvektoren, die die selbe Richtung hätten erkennen müssen, gemittelt, sodass zehn Vektoren mit beispielsweise sieben Neuronen letztlich einen Vektor mit sieben Elementen ergab. Lag das Maximum nun beispielsweise beim zweiten Element des Vektors, so wurde die entsprechende Richtung, die dem zweiten Element zugeordnet wurde vom Netzwerk klassifiziert. Liegen 30 Vektoren einer Richtung in der Ausgangs- bzw. Targetmatrix vor, sind entsprechend drei Entscheidungen des neuronalen Netzes vorhanden. Entfallen alle auf die richtige Richtung, so erscheint in der Confusion-Matrix ein Eintrag von 100%. Das ist die Hit-Rate für eine Richtung. Die gemittelte Hit-Rate meint das arithmetische Mittel über die diagonalen Einträge der Confusion-Matrix. Anders ausgedrückt ist die mittlere Hit-Rate das Mittel der Hit-Raten für jede einzelne Richtung. Die Codierung der Targetvektoren ist in Abschnitt 3.7 beschrieben.

## 4.4 Ergebnisse und Diskussion der Vorstudie

Die letzte Spalte der Tabelle 4.2 enthält die erreichten mittleren Hit-Raten. Die Confusion-Matrizen einzelner Fälle der Vorstudie sind in Abbildung 4.2 gezeigt. In der Horizontalebene sind die mittleren Hit-Raten, wie erwartet werden konnte, hoch für ILD und ITD, sowie für die Kombination daraus (vgl. Abb. 4.2 a, b). Ein noch besseres Ergebnis könnte man mit längeren Zeitfenstern für die ILD- und ITD-Berechnung oder durch ein Zeit-rekursives Glätten der Werte erreichen.

Die Kombination aus Betragsspektrum und Phase führte zu keiner Richtungserkennung (vgl. Abb. 4.2 c). Möglicherweise wären andere Parameter für das Netzwerk erforderlich gewesen. Im Gegensatz dazu konnten die binauralen Betragsspektren und die Kombination aus Real- und Imaginärteil beste Hit-Raten erreichen (vgl. Abb. 4.2 b). Dass DNN hat demnach die Informationen der Horizontalebene aus den Trainingsdaten extrahiert.

Soll das neuronale Netz Winkel bzw. einzelne Richtungen der Medianebene erkennen, erwartet man, dass monaurale Färbungen im Spektrum extrahiert werden. Diese treten bei Änderung der Elevation auf. Die perfekten Ergebnisse bei Verwendung der binauralen Betragsspektren lassen den Schluss zu, dass das DNN die Unterschiede der spektralen Färbung gelernt hat (vgl. Abb. 4.2 e).

Die ILD sollte für alle Winkel der Medianebene bei perfekter Ausrichtung des symmetrischen Kunstkopfes sowie der Mikrofone am Ohrkanal keine Änderung zeigen. Dennoch ergaben die Merkmale ILD und die Kombination aus ILD und ITD sehr gute Werte in der Medianebene (vgl. Abb. 4.2 d, e). Es ist wahrscheinlich, dass das Netzwerk kleinere Abweichungen bzw. Messungenauigkeiten bei der Erstellung des HRTF-Datensatzes innerhalb einzelner Bänder entdeckte und anhand dieser klassifizierte. Auch die Werte für die ITD ist oberhalb der Schätzrate (vgl. Abb. 4.2 f). Eine kleine aber systematische Abweichung des Azimutwinkels  $\vartheta$  über die Elevation könnte die Ursache für eine sich ändernde Charakteristik der ILD und ITD sein. Die Abweichung der ITD beträgt ungefähr  $40 \mu\text{s}$  verteilt über den Bereich verwendeter Elevationen von  $-10$  bis  $50^\circ$ . Das könnte von einer minimalen Abweichung beim Messaufbau herrühren. Eine andere Erklärung für das präzise Lokalisieren innerhalb der Medianebene mittels ILD bezieht sich auf die Richtungscharakteristik der Mikrophone, die während der HRTF-Messung verwendet wurden. Ohne eine omnidirektionale Charakteristik bleiben Änderungen in der ILD wahrscheinlich, falls Mikrophone in ihrer Orientierung für einzelne Messungen zunächst gleich gehalten und dann zur Referenzmessung verändert worden sind.<sup>3</sup> Im Falle dass die räumliche Orientierung des Mikrophones von eigentlicher Messung hin zur Referenzmessung geändert wurde, kann dieser Effekt nicht als Fehlerquelle eliminiert werden, was die Lokalisationsleistung des DNN bereits beeinflussen könnte.

Zur Lokalisierung der Entfernung kann die Absorption durch die Atmosphäre

---

<sup>3</sup>Es wird ein Referenzsignal ohne KEMAR Mannequin im Punkt des interauralen Zentrums aufgenommen. Dieses dient zur Entfaltung aufgenommener Signale um die reine HRIR bzw. HRTF zu erhalten. Dadurch sollten Anregungssignal, Fehlereffekte wie Reflexionen an Wänden und Einfluss des Lautsprechers entfernt werden (vgl. [Mø92]).

bzw. die Ausbreitungsdämpfung nicht für Sprache herangezogen werden. Ein Sprachspektrum hat nur wenige Anteile oberhalb von 10 kHz. Die verwendeten Logatome haben eine Abtastfrequenz von nur 16 kHz, enthalten demnach nur Frequenzanteile bis 8000 Hz. Die Ausbreitungsdämpfung ist vor allem für sehr hohe Frequenzen wirksam. Möglicherweise kann sie als Merkmal zur Detektion bei breitbandigen akustischen Ereignissen interessant sein. Wie in Abschnitt 3.5 beschrieben wurden Merkmalsvektoren zum Trainieren und Testen bezüglich einer Lokalisation in der räumlichen Tiefe angelegt. Mit dem Ergebnis der Vorstudie kommt man zu dem Schluss, dass das neuronale Netz in der Lage ist Entfernungen zu klassifizieren für Vektoren im Bereich von 18 - 22.05 kHz (vgl. Abb. 4.2 i). Die Confusion-Matrix zeigt allerdings eine fehleranfällige Schätzung. Es ist anzunehmen, dass für höhere Frequenzen die Performance des Netzes besser würde, da die Ausbreitungsdämpfung zu hohen Frequenzen hin zunimmt. Tiefere Frequenzbereiche wurden ebenfalls getestet, erreichten hingegen aber nur Hit-Raten knapp oberhalb der Schätzrate, da Dämpfungskoeffizienten für tiefe Frequenzen kleiner werden.

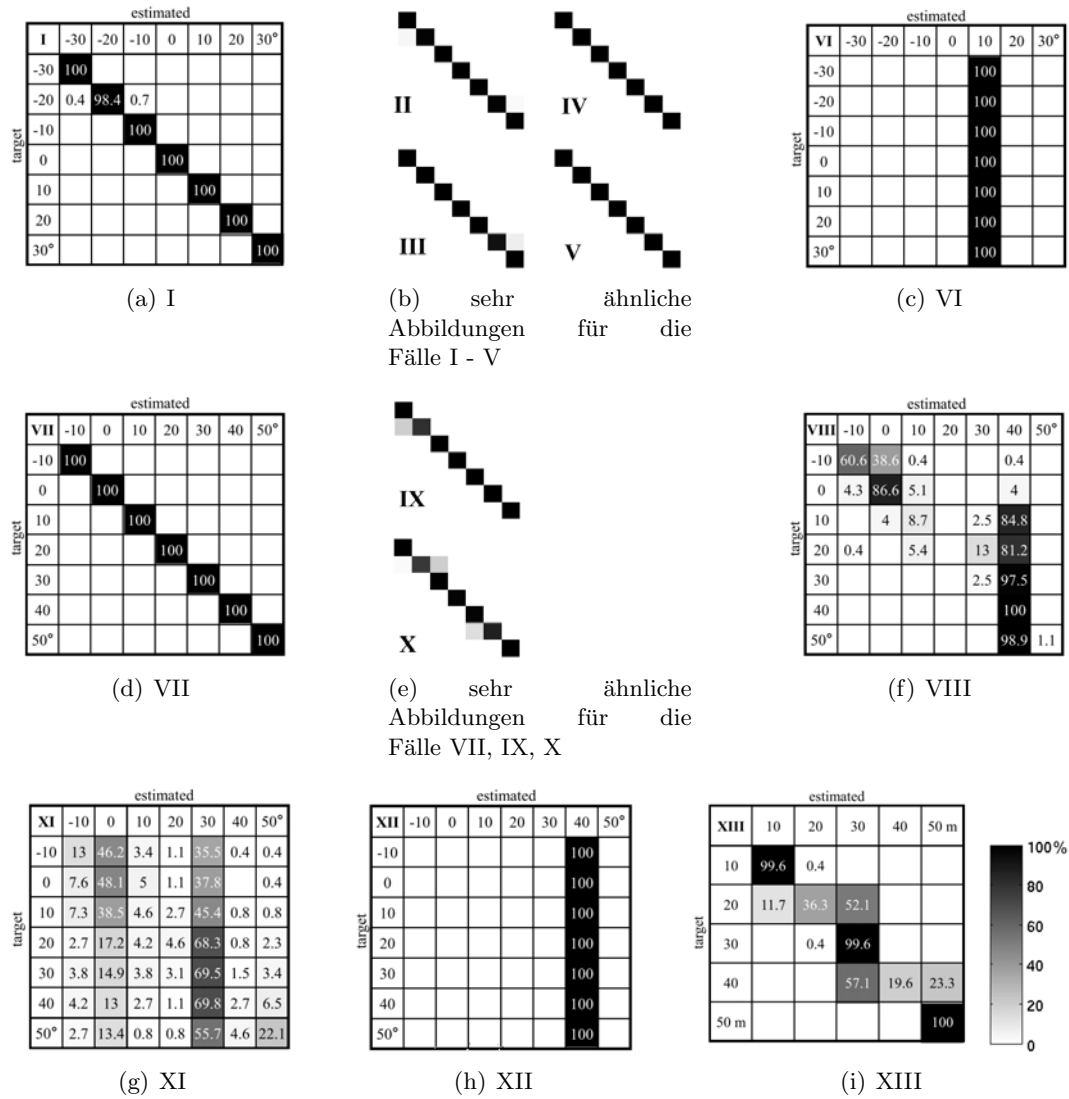


Abbildung 4.2: DNN-Performance der Vorstudie.

## Kapitel 5

# Sphärische Lokalisation

In einem weiteren Teil wurden zwei umfassendere **Fensterausschnitte** der Kugeloberfläche für **Sprache und Rauschen** betrachtet. Es handelt sich demnach um eine zweidimensionale Lokalisation von Azimuth und Elevation. Wie in Abbildung 3.1) zu sehen ist, wird ein frontales und ein seitliches Fenster mit jeweils  $7 \cdot 7$  Punkten gewählt.

Untersuchte **Merkmale** sind die ILD und ITD einzelner auditorischer ERB-Bänder, die Kombination aus ILD und ITD, das Betragsspektrum und im Weiteren die Mel-Frequenz-Cepstrum-Koeffizienten. Entsprechend der **Targetvektoren**, wie sie im Abschnitt 3.7 aufgeführt sind, wurden als Ausgangsvektoren drei verschiedene Varianten gewählt. Zum einen werden Azimuth und Elevation *parallel* in zwei Netzen unabhängig detektiert. Beide Netze verfügen im Training dabei über den gesamten Datensatz, erstellt mit 49 verschiedenen HRTFs je Fensterausschnitt bzw. über die 15 auf den Hauptachsen liegenden HRTFs für den Fall zur Überprüfung der Dimensionsabhängigkeit der Merkmale, wie im Weiteren noch beschrieben wird. Ausgangsvektoren haben jeweils die Länge von sieben Neuronen, stehend für die sieben Winkel jeder Dimension. Dazu wurden entsprechend zwei Targetvektoren mit Nullen und an codierter Stelle mit Eintrag „eins“ generiert. Zur zweiten Variante hat der Ausgangsvektor 14 Neuronen, da die sieben Azimuth- und sieben Elevationswinkel einfach *aneinander gehangen* wurden. Der Targetvektor erhält bei dieser Variante zwei Einträge mit einer eins, die anderen zwölf sind null. Zum Dritten wird ein großer Vektor von 49 Neuronen in *sphärischer* Darstellung als Ausgangsvektor gewählt. Ein Neuron steht dabei für einen Punkt im Koordinatensystem bzw. auf dem Ausschnitt der Kugeloberfläche. Der dazugehörige Targetvektor erhält an treffender Stelle den Eintrag „eins“. Die Codierung der Punkte bzw. der Elevations- und Azimuthinformation im Targetvektor ist willkürlich festgelegt und wird bei der Auswertung wieder berücksichtigt.

Neuronale Netze mit **Bottleneck** sind im Training aufwendiger wie in Abschnitt 3.8.2 beschrieben worden ist. Während ohne Bottleneck das Netzwerk für drei versteckte Ebenen ebenenweise in vier Teilen trainiert wurde, sind für das Training mit Bottleneck zunächst mit vier Teilen ebenenweise „Bottleneck-Features“ ermittelt worden, ähnlich einer Hauptkomponentenanalyse, um mit diesem Vortraining weitere vier Teildurchgänge das Netzwerk ohne Bottleneck zu trainieren. Entsprechend ist der Trainingsaufwand mit Bottleneck hier ungefähr doppelt so hoch.

Fett hervorgehoben sind in diesem Abschnitt die Faktoren dieses Teils der Arbeit.

## 5.1 Netzkonditionen

Die Struktur der Netze sind weitgehend aus der Vorstudie übernommen worden. Es handelt sich nun allerdings um fünf-schichtige Netze. Das Momentum ist zum Wert 0.04 und die Lernrate  $\eta$  zu 0.5 festgelegt worden. Das Stopkriterium für die Größe des mittleren quadratischen Fehlers ist bei einem Wert von  $10^{-8}$  geblieben.

## 5.2 Untersuchung der Dimensionsunabhängigkeit

Zunächst wird für den frontalen Fall bei Rauschen untersucht, ob das Netzwerk in der Lage ist aus den Informationen der Hauptachsen auf Punkte der Kugeloberfläche zu schließen. Im Training des neuronalen Netzes werden demnach nur Eingangsvektoren verwendet, deren zugehörige Winkel das reduzierte Grid wie in Abbildung 4.1 darstellen. Beim Testen des Netzwerkes sind jedoch alle anderen möglichen Winkelkombinationen, insgesamt 49 (7·7), wie in Abbildung 3.1 enthalten. Selbes Vorgehen findet sich in der Studie von Neti et al. 1992 [NYS92]. Wäre das Netzwerk in der Lage dazu von dem reduzierten Grid auf weitere Punkte schließen zu können, wäre von einem dimensionsunabhängigen Merkmal auszugehen. Mit unabhängig ist gemeint, dass sich ein Merkmal über den Azimuth unabhängig von der Elevation verändert und umgekehrt.

Zu dieser Untersuchung wurden wie erwähnt Merkmale extrahiert aus mit HRTF gefalteten Rauschsignalen. Netzkonditionen sind die im vorangegangenen Abschnitt 5.1 Beschriebenen. Zur Auswertung der Daten wurde auch hier der Vergleich der mittleren Hit-Raten angestellt mit der in Abschnitt 4.3 geschilderten Berechnung bzw. Vorgehensweise.

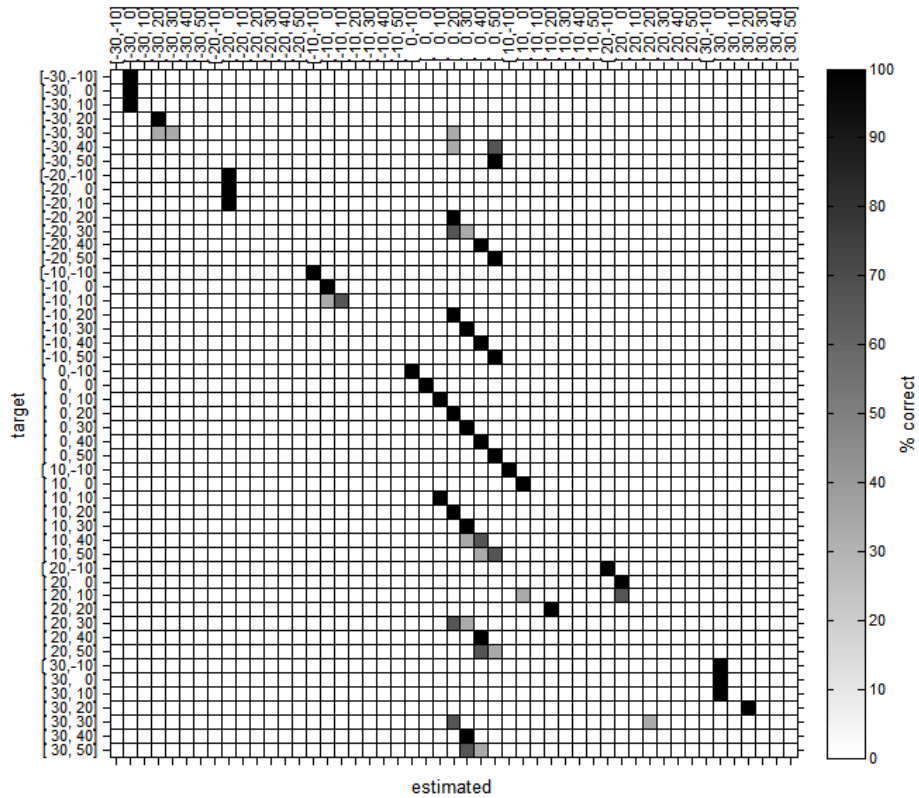
Tabelle 5.1: mittlere Hit-Raten der Untersuchung zur Dimensionsunabhängigkeit für frontale HRTF mit Rauschen. Werte für untersuchte Merkmal finden sich je Art des Targetvektors mit oder ohne Bottleneck

|           | bottle   |           |           | no bottle |           |           |
|-----------|----------|-----------|-----------|-----------|-----------|-----------|
|           | parallel | angehängt | sphärisch | parallel  | angehängt | sphärisch |
| ILD ITD   | 26.53    | 26.53     | 26.53     | 26.53     | 26.53     | 26.53     |
| ILD       | 26.53    | 26.53     | 26.53     | 26.53     | 26.53     | 26.53     |
| ITD       | 18.37    | 18.37     | 18.37     | 18.37     | 18.37     | 18.37     |
| magnitude | 26.53    | 26.53     | 26.53     | 38.78     | 27.89     | 26.53     |
| mfcc      | 28.57    | 26.53     | 26.53     | 31.29     | 26.53     | 26.53     |

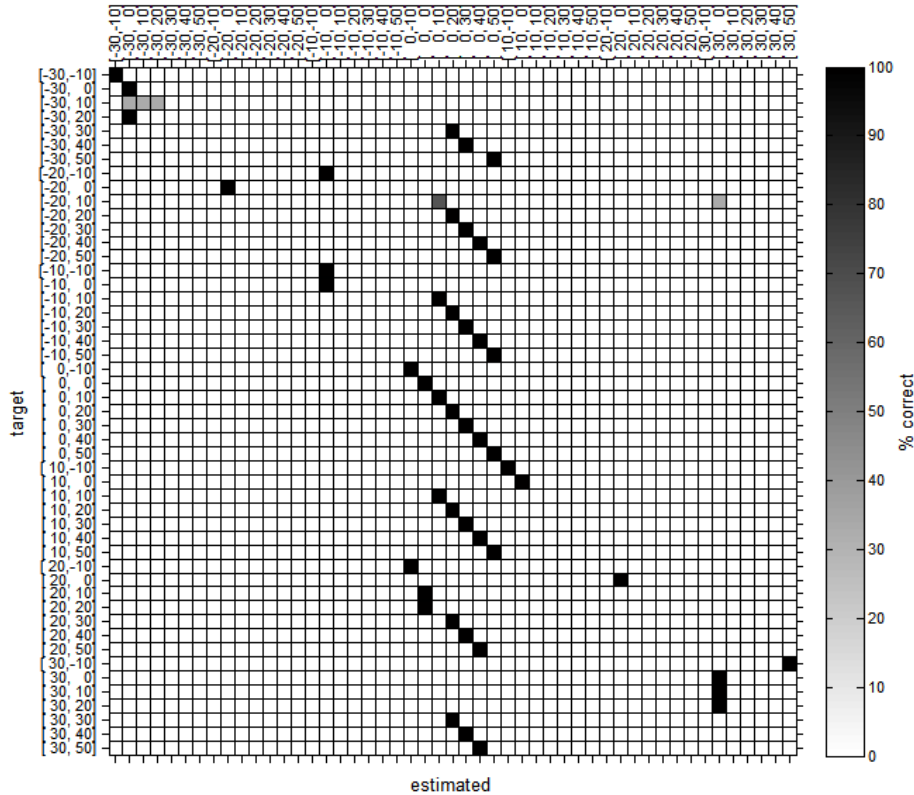
Ob mit oder ohne Bottleneck und ungeachtet der Art des Ausgangs- bzw. Targetvektors, deuten die Werte der Tabelle 5.1 für die mittleren Hit-Raten dieser Untersuchung darauf hin, dass es sich um dimensionsabhängige Merkmale handelt. Es ist demnach von Nöten dem Netzwerk für das Training den gesamten Datensatz aller Winkel zur Verfügung zu stellen. Auf die Darstellung aller



Confusion-Matrizen soll an dieser Stelle verzichtet werden. Hier sollen lediglich zwei Beispiele aufgezeigt werden (siehe Abb. 5.2). Die beiden Grafiken zeigen die zwei besten Ergebnisse für diese Betrachtung. Es wurden bei parallelem Training ohne Bottleneck für das Betragsspektrum eine mittlere Hit-Rate von 38.7% und für das Merkmal der Cepstral-Koeffizienten 31.29% erreicht (vgl. Tab. 5.1). In den zugehörigen Abbildungen ist zu erkennen, dass das Netzwerk viele Punkte verwechselt. Für beide Fälle lokalisiert das Netzwerk überproportional viele Punkte für Azimuth  $\vartheta = 0^\circ$ . Die Elevation wird dabei aber häufiger richtig zugeordnet. In der Abbildungen zeigt sich das als vertikal angeordnete Parallelen. Möchte man erklären, dass mittlere Hit-Raten auch für andere Fälle zur Untersuchung der Dimensionsunabhängigkeit oberhalb der Schätzrate liegen, wird sich das noch in der weiteren Analyse zur Eignung einzelner Merkmale zeigen. Denn Punkte der Kugeloberfläche die im Trainingsdatensatz enthalten waren und auch zum Teil benachbarte Punkte, wurden vom Netzwerk erkannt aufgrund einer guten Eignung verwendeter Merkmale.



(a) für das Merkmal des Betragsspektrums



(b) für das Merkmal der Mel-Frequenz-Cepstrum-Koeffizienten

Abbildung 5.1: Ausgewählte Confusion-Matrizen zur Untersuchung der Dimensionsunabhängigkeit. Netzwerk ohne Bottleneck, frontaler Fall, Azimuth und Elevation wurden parallel trainiert, für Rauschen.

### 5.3 Dimensionsabhängige Untersuchung

Da hier gewählte Merkmale dimensionsabhängig zu betrachten sind, folgt das weitere Training der neuronalen Netze mit jeweils vollständigem Datensatz aller 7·7 Punkte der Fensterausschnitte. Auch hier wurden Netzkonditionen wie in Abschnitt 5.1 notiert belassen. Es wurden neuronale Netze für Rauschen und Sprache trainiert, für genannte Merkmale und auch hier für zwei Arten des Netzes mit und ohne Bottleneck.

Tabellen 5.2 und 5.3 zeigen die Werte für Rauschen und Sprache getrennt von einander. Innerhalb der Tabellen werden jeweils die mittleren Hit-Raten einzelner Merkmale für das neuronale Netz ohne und mit Bottleneck gelistet mit den Unterkategorien des Fensterausschnittes auf der Kugeloberfläche (seitlich, frontal) und der Art des Targetvektors (parallel, angehängt, sphärisch) wie auch schon im vorangegangenen Teil.

Es folgt die qualitative Auswertung dieser Tabellen in Abschnitten zu aufgeführten fünf Faktoren der Untersuchung. Confusion-Matrizen finden sich im Anhang A.

Tabelle 5.2: mittlere Hit-Rate getesteter Netzkonditionen für Rauschen

| front |           | bottle   |           |           | no bottle |           |           |
|-------|-----------|----------|-----------|-----------|-----------|-----------|-----------|
| noise |           | parallel | angehängt | sphärisch | parallel  | angehängt | sphärisch |
|       | ILD ITD   | 44.90    | 34.69     | 44.90     | 32.65     | 42.86     | 51.02     |
|       | ILD       | 100.00   | 100.00    | 100.00    | 100.00    | 73.47     | 100.00    |
|       | ITD       | 28.57    | 34.69     | 34.69     | 26.53     | 20.41     | 34.69     |
|       | magnitude | 98.64    | 98.64     | 100.00    | 81.63     | 2.04      | 90.48     |
|       | mfcc      | 100.00   | 100.00    | 100.00    | 100.00    | 100.00    | 100.00    |
|       |           |          |           |           |           |           |           |
| side  |           | bottle   |           |           | no bottle |           |           |
| noise |           | parallel | angehängt | sphärisch | parallel  | angehängt | sphärisch |
|       | ILD ITD   | 4.08     | 8.16      | 4.08      | 8.16      | 4.08      | 4.08      |
|       | ILD       | 6.12     | 10.20     | 10.20     | 8.16      | 4.08      | 10.20     |
|       | ITD       | 4.08     | 4.08      | 6.12      | 4.08      | 4.08      | 4.08      |
|       | magnitude | 95.92    | 94.56     | 96.60     | 45.58     | 21.77     | 80.27     |
|       | mfcc      | 100.00   | 100.00    | 100.00    | 100.00    | 100.00    | 100.00    |

Tabelle 5.3: mittlere Hit-Rate getesteter Netzkonditionen für Sprache

| front  |           | bottle   |           |           | no bottle |           |           |
|--------|-----------|----------|-----------|-----------|-----------|-----------|-----------|
| speech |           | parallel | angehängt | sphärisch | parallel  | angehängt | sphärisch |
|        | ILD ITD   | 56.09    | 56.96     | 60.23     | 59.98     | 53.62     | 55.89     |
|        | ILD       | 99.10    | 97.40     | 98.84     | 98.65     | 96.08     | 97.83     |
|        | ITD       | 29.44    | 30.10     | 32.74     | 31.53     | 33.54     | 35.23     |
|        | magnitude | 99.59    | 99.50     | 99.75     | 79.24     | 88.03     | 89.15     |
|        | mfcc      | 98.28    | 97.69     | 97.75     | 98.32     | 98.19     | 99.35     |
|        |           |          |           |           |           |           |           |
| side   |           | bottle   |           |           | no bottle |           |           |
| speech |           | parallel | angehängt | sphärisch | parallel  | angehängt | sphärisch |
|        | ILD ITD   | 4.72     | 7.56      | 8.29      | 5.45      | 5.68      | 7.43      |
|        | ILD       | 89.64    | 88.78     | 88.22     | 78.61     | 74.58     | 84.14     |
|        | ITD       | 3.37     | 4.49      | 3.07      | 4.49      | 4.52      | 5.90      |
|        | magnitude | 99.93    | 99.93     | 99.88     | 95.13     | 99.44     | 84.55     |
|        | mfcc      | 99.21    | 97.85     | 99.57     | 99.62     | 99.28     | 99.63     |

### 5.3.1 Vergleich Rausch- und Sprachsignale

Rot unterlegte Zahlen der Tabellen 5.2 und 5.3 gehören zu nicht interpretierbaren Matrizen. Für diese Werte hat das Netz keinerlei Muster gelernt. Auch wenn mittlere Hit-Raten höher als die Schätzrate liegen, ist das Bild der Confusion Matrix unsinnig. Diese Konditionen eignen sich demnach nicht zur Lokalisierung der Audiosignale.

Bei qualitativer Betrachtung der Tabellen für Sprache und Rauschen fällt auf, dass für Sprache mehr Ergebnisse interpretierbar sind, mittlere Hit-Raten oberhalb der Schätzrate liegen und zudem Confusion-Matrizen in ihrer Abbildung auf mehr oder weniger erfolgreiche Mustererkennung des Netzes hindeuten. Das beschreibt einen Trend, den man nicht ohne Weiteres erwarten würde, da eine Sprachdatei eines Logatoms mehr Variabilität aufweist als eine Audiodatei selber Länge mit rosa Rauschen. Im umgekehrten Fall lassen sich für Rauschen viele mittlere Hit-Raten mit dem Wert 100% zählen; für Sprache gibt es keinen Eintrag mit diesem Wert. An dieser Stelle ohne quantitative Betrachtung würde man eher der Erwartung folgen und generell die besseren Ergebnisse für Rauschen vermuten.

### 5.3.2 Vergleich frontaler und seitlicher Lokalisation

Das richtungsgenaue Lokalisieren ist bei frontalem Schalleinfall für das menschliche Gehör einfacher als für seitliche Schallquellen. Merkmale sind über den gleichen Winkel hinweg differenzierter und führen zu einer besseren Lokalisationsleistung. Entsprechend war die Intention neben einer üblichen frontalen Situation mit besseren zu erwartenden Ergebnissen das neuronale Netz auch für einen eher konservativen Fall zu trainieren. *Für Rauschen* scheint sich dieser Trend bei Merkmalen der ILD und ITD zu bestätigen. Während eine noch recht gute Leistung für eine frontale Schallquelle erzielt werden konnte, kommt es zu nicht interpretierbaren und eher zufälligen Ergebnissen bei seitlichem Schalleinfall. Für das Betragsspektrum mit vortrainierten Bottleneck-Merkmalen bei Rauschen sind Werte der mittleren Hit-Rate alle kurz unterhalb von 100% angesiedelt. Einen Unterschied der Lokalisationsleistung in Abhängigkeit des Fensterausschnittes ist nicht zu beobachten. Beim Merkmal der MFCC lässt ein Deckeneffekt für Rauschen kein Urteil zu. Hier wird die maximale Lokalisationsleistung erreicht. *Bei Sprache* erreicht das Netz für Cepstral-Koeffizienten nicht die beste Leistung, Werte sind hier aber für den frontalen und seitlichen Fall sehr ähnlich. Das ist jedoch nur ein qualitatives Urteil. Möglicherweise sind aufgezeigte Unterschiede nicht signifikant. Für Sprachsignale zeichnet sich für die ILD und ITD und deren Kombination die gleiche Tendenz ab. Frontal werden zum Teil sehr gute Ergebnisse erreicht. Seitliche Richtungsunterscheidung scheint dem Netzwerk weniger zu gelingen. Für die Merkmale MFCC und des Betragsspektrums gibt es Trainingsdurchläufe, die mal für das frontale und mal für das seitliche HRTF-Grid besser ausfielen. Man kann vorsichtig annehmen, dass das neuronale Netz ähnlich dem menschlichen Gehör für ILD und ITD ungenügend Informationen für eine sichere seitliche Lokalisation erhielt. Für das Betragsspektrum und die Cepstralkoeffizienten scheint dies nicht zuzutreffen. Das neuronale Netz unterscheidet auch für den seitlichen Fall Richtungen gut. Vorn-Hinten-Vertauschungen können nicht beobachtet werden. Für genaue Aussagen müsste das Netz häufiger trainiert werden und vorhandene Ergebnisse dann einer quantitativen Analyse zugeführt werden.

### 5.3.3 Lohnt der Mehraufwand des Bottlenecks?

Für Rauschen ergeben sich bessere Hit-Raten für das Bottleneck. Grün unterlegte Zahlen sind ungefähr zehn und mehr Prozentpunkte höher als das äquivalente Ergebnis ohne Bottleneck. Für Sprache schwindet der Unterschied. Nur noch drei Werte für das Merkmal des Betragsspektrums scheinen das aufwendigere Training

des neuronalen Netzes mit Bottleneck zu rechtfertigen. Jedoch ist dieser Effekt nur für das frontale Fenster zu beobachten. Seitlich schwindet dieser Vorteil, da auch hier für den Fall ohne Bottleneck beim Betragsspektrum recht gute Ergebnisse erreicht worden sind. Man muss schlussfolgern, dass sich für Rauschen der Mehraufwand uneingeschränkt lohnt. Für Sprachsignale ist die genauere Betrachtung der Merkmale nötig.

#### 5.3.4 Eignung der Targetvektoren

Es wurden verschiedene Targetvektoren untersucht, die einen unterschiedlich hohen Aufwand im Training bedeuten. Mit Ergebnissen aus obenstehenden Tabellen scheint keine Variante sich von den anderen konsistent abzuheben. Daten müssen mit Sorgfalt interpretiert werden, da es sich um jeweils ein Trainingsdurchgang handelt. Abweichungen von diesen Ergebnissen sind für einen weiteren Trainingsdurchgang denkbar. So sollte der Trainingsdurchgang der Kondition des Betragsspektrums bei frontalen Rauschen ohne Bottleneck und dem Ausgangsvektor für aneinanderghängte Azimuth- und Elevationsvektoren noch einmal untersucht werden. Möglicherweise führte der Gradientenabstieg zu einem lokalen statt dem globalen Minimum der Fehlerfunktion.

#### 5.3.5 Auswertung zur Wahl der Merkmale

Da das neuronale Netz wünschenswerter Weise omnidirektional eingesetzt werden soll, sind Betragsspektrum und MFCC die bessere Wahl unter den Merkmalen. Diese beiden Merkmale weisen durchgehend gute Ergebnisse auf, mit einer Ausnahme wie im vorherigen Abschnitt beschrieben worden ist. Mittels der ILD können noch ebenso gute Ergebnisse für den frontalen Fall erzielt werden. Bemerkenswerter Weise sind Werte der mittleren Hit-Rate über alle Konditionen mit seitlichem Schalleinfall bei Sprache zwischen 74.58% und 89.64% konsistent im akzeptablen Regionen. Hingegen sind alle Fälle der ILD mit seitlichem Schalleinfall bei Rauschen nicht interpretierbar. Dieser Effekt scheint überzufällig.

## Kapitel 6

# Fazit - Ausblick

In dieser Masterarbeit konnten neuronale Netze trainiert werden, Schallquellen Richtungen zuzuweisen. Anhand einer Vorstudie wurden zunächst Skripte in Matlab für die OXlearn-Toolbox entwickelt, die ein Training mehrschichtiger Netze mit dem Backpropagation-Algorithmus ermöglichen. In einer explorativen Vorstudie wurden geeignete Parameter des Netzes gefunden, um zunächst Winkel der einzelnen Dimensionen für das neuronale Netz zu detektieren. Ebenfalls wurden in der Vorstudie verschiedene Eingangsvektoren des neuronalen Netzes auf ihre Eignung geprüft. Zum Teil bestätigte sich die Eignung dieser. Passend erwiesen sich die ILD, ITD und das Betragsspektrum, die als Merkmale aus mit HRTFs gefalteten Logatomen vorbereitend extrahiert wurden.

Für die Lokalisation der räumlichen Tiefe konnten keine zufriedenstellenden Ansätze aufgetan werden. Wie zu erwarten war, zeigte das Netzwerk nach Simulation der Ausbreitungsdämpfung besonders für das Training mit hochfrequentem Rauschen bessere Ergebnisse, da im hohen Frequenzbereich die Ausbreitungsdämpfung erst wirksam wird. Bei den Ergebnissen kann man jedoch nur von einer tendenziellen Tiefenlokalisation sprechen. Mit zusätzlichen Störeffekten in realistischer Umgebung würde auch diese Leistung sicher nicht wiederholbar sein.

In einem weiteren Teil wurde mit angepassten Netzwerkparametern und einem zusätzlichen Vortraining mittels Bottleneck-Architektur die sphärische Lokalisation angestrebt. Für frontale Richtungen wurden zunächst Merkmale dieses Studienteils auf ihre Dimensionsabhängigkeit überprüft, indem das Netz mit den räumlichen Informationen der Horizontal- und Medianebene trainiert auf Richtungen neben den Hauptachsen liegend schließen sollte. Das gelang bei keinem Merkmal. Demnach sind Netze für folgende Untersuchungen mit dem vollständigen Datensatz trainiert worden.

Die weiteren Betrachtungen schlossen das Verhalten des neuronalen Netzes bei seitlichem Schalleinfall ein. Mit einem Vortraining zur Reduktion des Eingangs auf Bottleneck-Merkmale konnte zu dem Schluss gekommen werden, dass seitliche und frontale Lokalisation sich für diesen Fall in ihrer Güte qualitativ betrachtet nicht unterscheiden. Das Vortraining mit Bottleneck bestätigte sich als vorteilhaft.

Generell eigneten sich Logatome als Sprachmaterial und erzielten für das Merkmal MFCC und das Betragsspektrum Ergebnisse nur wenig unter den Ergebnissen für Rauschen; teils wurden sogar bessere Werte erzielt. Das Betragsspektrum und vor allem die Mel-Frequenz-Cepstrum-Koeffizienten bewährten sich bei ausgezeichneten

Lokalisationsleistungen bei gegebener HRTF-Auflösung. Interessant sind weitere Trainingsdurchläufe für eine feinere Auflösung der Richtungen mit vorheriger Prüfung der Interpolationsfähigkeit des neuronalen Netzes.

Zur Wahl des Targetvektors ist zu einer parallelen Schätzung von Azimuth und Elevation zu raten, da ein paralleles Training und einhergehende kleinere Ausgangsvektoren Zeit ersparen. Für Rauschen genügte bei gleicher Epochenzahl im Training ein kleinerer Datensatz, um sehr gute Richtungsschätzungen zu erzielen. Das Training mit Logatomen hingegen war sehr zeitintensiv. Es ist bei gewähltem Algorithmus leider nicht auszuschließen, dass der Gradientenabstieg nicht zum globalen Minimum der Fehlerfunktion führt, sondern zu einem lokalen Minimum. Dann muss erneut trainiert werden bei eventueller Anpassung der Netzparameter. Welche Cues innerhalb der extrahierten Merkmale tatsächlich dem neuronalen Netz zur Lokalisation dienen kann letztlich auch nicht beantwortet werden, was einen Nachteil für weitere Analysen darstellt.

Ergebnisse dieser Arbeit sprechen dennoch für eine erfolgreiche Lokalisation mittels DNN für hier gewählte Konditionen und machen weitere Untersuchungen für höhere HRTF-Auflösungen lohnenswert.

# Literaturverzeichnis

- [ADMT01] V Ralph Algazi, Richard O Duda, Reed P Morrison, and Dennis M Thompson. Structural composition and decomposition of hrtfs. In *Applications of Signal Processing to Audio and Acoustics, 2001 IEEE Workshop on the*, pages 103–106. IEEE, 2001. (Cited on p. 12.)
- [BB08] Jens Blauert and Jonas Braasch. Räumliches Hören. In Stefan Weinzierl, editor, *Handbuch der Audiotechnik*, page 87–121. Springer, 2008. (Cited on pp. 1, 4.)
- [Bla97] Jens Blauert. *Spatial hearing*. Cambridge, MIT Press, 1997. (Cited on p. 1.)
- [BLW<sup>+</sup>13] Fabian Brinkmann, Alexander Lindau, Stefan Weinzierl, Gunnar Geissler, and Steven van de Par. A high resolution head-related transfer function database including different orientations of head above the torso. In *Proceedings of the AIA-DAGA 2013 Conference on Acoustics*, 2013. (Cited on p. 11.)
- [Czy03] Andrzej Czyzewski. Automatic identification of sound source position employing neural networks and rough sets. *Pattern Recognition Letters*, 24(6):921–933, 2003. (Cited on p. 1.)
- [DPM96] Michael S Datum, Francesco Palmieri, and Andrew Moiseff. An artificial neural network for sound localization using binaural cues. *The Journal of the Acoustical Society of America*, 100(1):372–383, 1996. (Cited on p. 1.)
- [GF08] František Grézl and Petr Fousek. Optimizing bottle-neck features for lvcsr. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 4729–4732. IEEE, 2008. (Cited on p. 8.)
- [GKKC07] Frantisek Grézl, Martin Karafiát, Stanislav Kontár, and Jan Cernocky. Probabilistic and bottle-neck features for lvcsr of meetings. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–757. IEEE, 2007. (Cited on p. 8.)
- [Hir06] Akira Hirose. *Complex-valued neural networks*. Springer, 2006. (Cited on p. 6.)



- [ISO99] DIN ISO. 9613-2 “dämpfung des schalls bei der ausbreitung im freien—allgemeines berechnungsverfahren “. *Ausgabe Oktober*, 1999. (Cited on p. 13.)
- [KBK<sup>+</sup>12] Rudolf Kruse, Christian Borgelt, Frank Klawonn, Christian Möwes, Georg Ruß, and Matthias Steinbrecher. *Computational Intelligence*. Springer-Verlag, 2012. (Cited on p. 6.)
- [Kra91] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991. (Cited on p. 8.)
- [Mø92] Henrik Møller. Fundamentals of binaural technology. *Applied Acoustics*, 36:171–218, 1992. (Cited on pp. 5, 22.)
- [MBK11] Bernd T Meyer, Thomas Brand, and Birger Kollmeier. Effect of speech-intrinsic variations on human and automatic recognition of spoken phonemes. *The Journal of the Acoustical Society of America*, 129(1):388–403, 2011. (Cited on p. 13.)
- [MJW<sup>+</sup>10] Bernd T Meyer, Tim Jürgens, Thorsten Wesker, Thomas Brand, and Birger Kollmeier. Human phoneme recognition depending on speech-intrinsic variability). *The Journal of the Acoustical Society of America*, 128(5):3126–3141, 2010. (Cited on p. 13.)
- [MvdPK11] Tobias May, Steven van de Par, and Armin Kohlrausch. A probabilistic model for robust localization based on a binaural auditory front-end. *Audio, Speech, and Language Processing, IEEE Transactions on*, 19(1):1–13, 2011. (Cited on p. 1.)
- [MZR08] Roland Mühler, Michael Ziese, and Dorothea Rostalski. Development of a speaker discrimination test for cochlear implant users based on the oldenburg logatome corpus. *ORL*, 71(1):14–20, 2008. (Cited on p. 13.)
- [Nic10] R Nicol. Binaural technology. In *Audio Engineering society*, 2010. (Cited on p. 4.)
- [NYS92] Chalapathy Neti, Eric D Young, and Michael H Schneider. Neural network models of sound localization based on directional filtering by the pinna. *The Journal of the Acoustical Society of America*, 92(6):3140–3156, 1992. (Cited on pp. 1, 2, 5, 26.)
- [RMG<sup>+</sup>15] Reinhild Roden, Niko Moritz, Stephan Gerlach, Stefan Weinzierl, and Stefan Goetze. On sound source localization of speech signals using deep neural networks. In *Proceedings of German Annual Conference on Acoustics (DAGA)*, Nuremberg, 2015. (Cited on p. 2.)
- [RW09] Nicolas Ruh and Gert Westermann. Oxlearn: A new matlab-based simulation tool for connectionist models. *Behavior research methods*, 41(4):1138–1143, 2009. (Cited on p. 17.)

- [SKKS09] Christina Stoica-Klüver, Jürgen Klüver, and Jörn Schmidt. *Modellierung komplexer Prozesse durch naturanaloge Verfahren*. Wiesbaden: Vieweg-Teubner, 2009. (Cited on p. 6.)
- [Sla93] Malcolm Slaney. An efficient implementation of the patterson-holdsworth auditory filter bank. *Apple Computer, Perception Group, Technical Report*, 1993. (Cited on p. 15.)
- [TvdP15] Joachim Thiemann and S van de Par. Multiple model high-spatial resolution hrtf measurements. In *Proc. DAGA*, 2015. (Cited on p. 11.)
- [WMW<sup>+</sup>05] Thorsten Wesker, Bernd T Meyer, Kirsten Wagener, Jörn Anemüller, Alfred Mertins, and Birger Kollmeier. Oldenburg logatome speech corpus (ollo) for speech recognition experiments with humans and machines. In *Interspeech*, pages 1273–1276, 2005. (Cited on p. 13.)
- [Yuh92] BP Yuhas. Automated sound localization through adaptation. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 2, pages 907–912. IEEE, 1992. (Cited on p. 2.)

# Abbildungsverzeichnis

|      |  |    |
|------|--|----|
| 2.1  | Skizze des Schallwegs zum Trommelfell . . . . .  | 5  |
| 2.2  | Schematische Darstellung eines mehrschichtigen neuronalen Netzes mit L versteckten Ebenen. . . . .                         | 7  |
| 2.3  | Schematische Darstellung eines mehrschichtigen neuronalen Netzes mit Bottleneck der letzten Ebene vor dem Ausgang. . . . . | 8  |
| 3.1  | Grid . . . . .   | 12 |
| 4.1  | Grid verwendeter HRIRs, $[\vartheta; \varphi]$ . . . . .   | 19 |
| 4.2  | DNN-Performance der Vorstudie. . . . .   | 24 |
| 5.1  | Ausgewählte Confusion-Matrizen zur Untersuchung der Dimensionsunabhängigkeit. . . . .                                      | 28 |
| A.1  | Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; frontal; parallel; . . . . .                 | 43 |
| A.2  | Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; frontal; angehängt; . . . . .                | 46 |
| A.3  | Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; frontal; sphärisch; . . . . .                | 49 |
| A.4  | Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; seitlich; parallel; . . . . .                | 52 |
| A.5  | Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; seitlich; angehängt; . . . . .               | 55 |
| A.6  | Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; seitlich; sphärisch; . . . . .               | 58 |
| A.7  | Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; frontal; parallel; . . . . .                | 61 |
| A.8  | Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; frontal; angehängt; . . . . .               | 64 |
| A.9  | Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; frontal; sphärisch; . . . . .               | 67 |
| A.10 | Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; seitlich; parallel; . . . . .               | 70 |
| A.11 | Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; seitlich; angehängt; . . . . .              | 73 |
| A.12 | Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; seitlich; sphärisch; . . . . .              | 76 |

|      |   |     |
|------|---|-----|
| A.13 | Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit<br>Bottleneck; frontal; parallel; . . . . .    | 79  |
| A.14 | Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit<br>Bottleneck; frontal; angehängt; . . . . .   | 82  |
| A.15 | Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit<br>Bottleneck; frontal; sphärisch; . . . . .   | 85  |
| A.16 | Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit<br>Bottleneck; seitlich; parallel; . . . . .   | 88  |
| A.17 | Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit<br>Bottleneck; seitlich; angehängt; . . . . .  | 91  |
| A.18 | Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit<br>Bottleneck; seitlich; sphärisch; . . . . .  | 94  |
| A.19 | Confusion-Matrizen für die Konditionen: Sprache; ohne<br>Bottleneck-Vortraining; frontal; parallel; . . . . .   | 97  |
| A.20 | Confusion-Matrizen für die Konditionen: Sprache; ohne<br>Bottleneck-Vortraining; frontal; angehängt; . . . . .  | 100 |
| A.21 | Confusion-Matrizen für die Konditionen: Sprache; ohne<br>Bottleneck-Vortraining; frontal; sphärisch; . . . . .  | 103 |
| A.22 | Confusion-Matrizen für die Konditionen: Sprache; ohne<br>Bottleneck-Vortraining; seitlich; parallel; . . . . .  | 106 |
| A.23 | Confusion-Matrizen für die Konditionen: Sprache; ohne<br>Bottleneck-Vortraining; seitlich; angehängt; . . . . . | 109 |
| A.24 | Confusion-Matrizen für die Konditionen: Sprache; ohne<br>Bottleneck-Vortraining; seitlich; sphärisch; . . . . . | 112 |

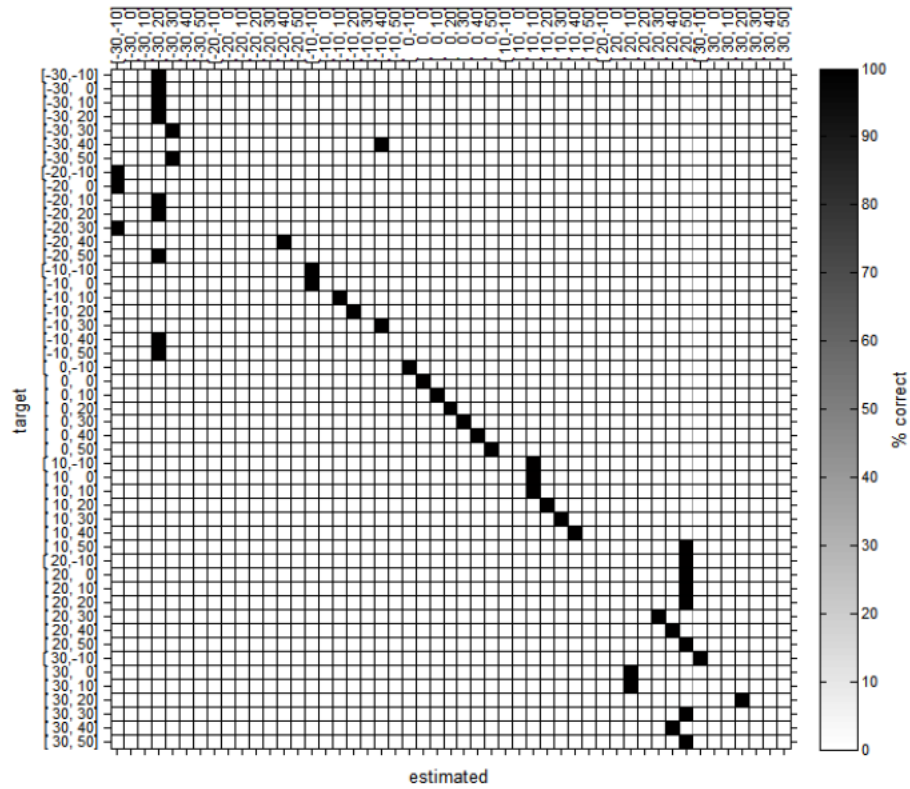
# Tabellenverzeichnis

|     |   |    |
|-----|---|----|
| 4.1 | Eingangsvektoren des DNN und deren Länge . . . . .                  | 20 |
| 4.2 | Getestete Fälle der Vorstudie . . . . .                             | 21 |
| 5.1 | Mittlere Hit-Raten der Untersuchung zur Dimensionsunabhängigkeit    | 26 |
| 5.2 | mittlere Hit-Rate getesteter Netzkonditionen für Rauschen . . . . . | 29 |
| 5.3 | mittlere Hit-Rate getesteter Netzkonditionen für Sprache . . . . .  | 29 |

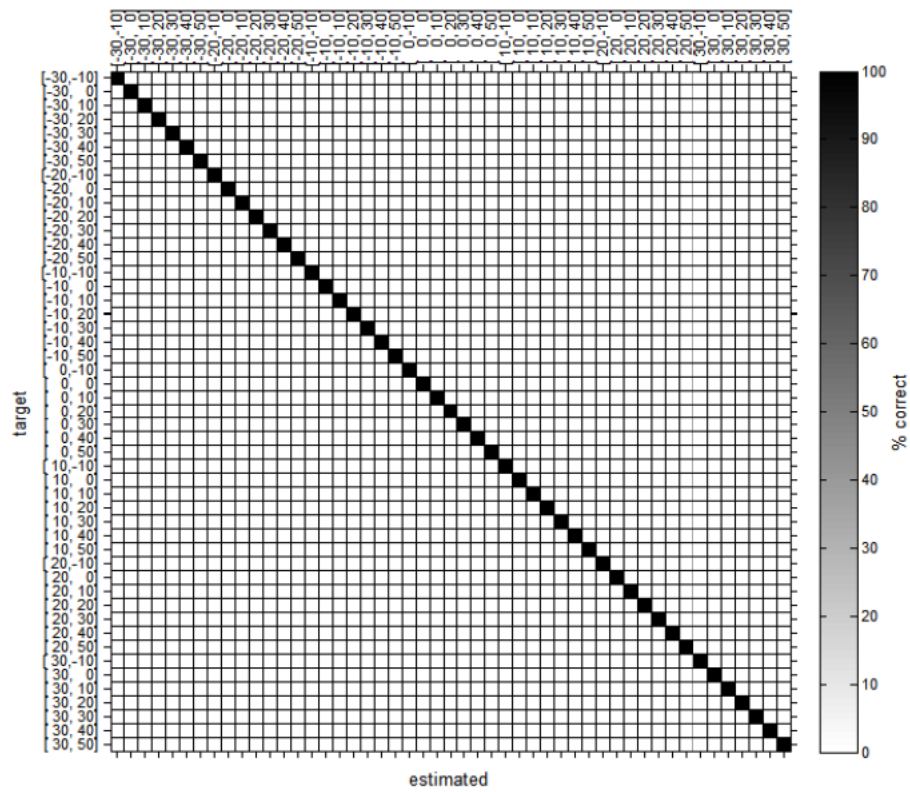
## Anhang A

# Abbildungen

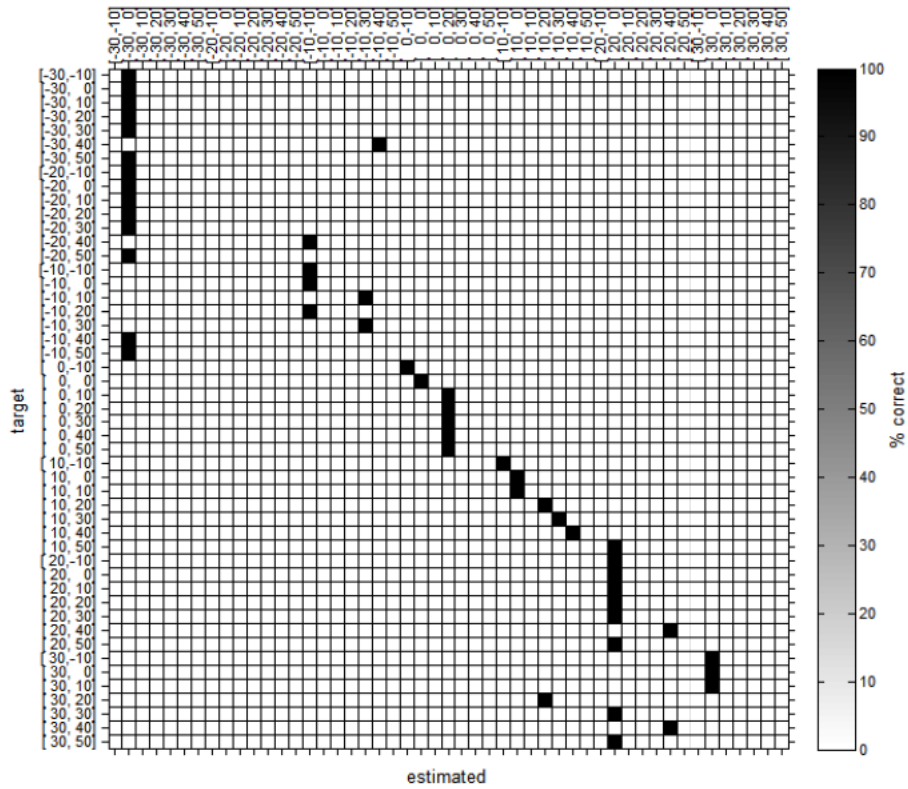
### i Confusion-Matrizen zur sphärischen Lokalisation



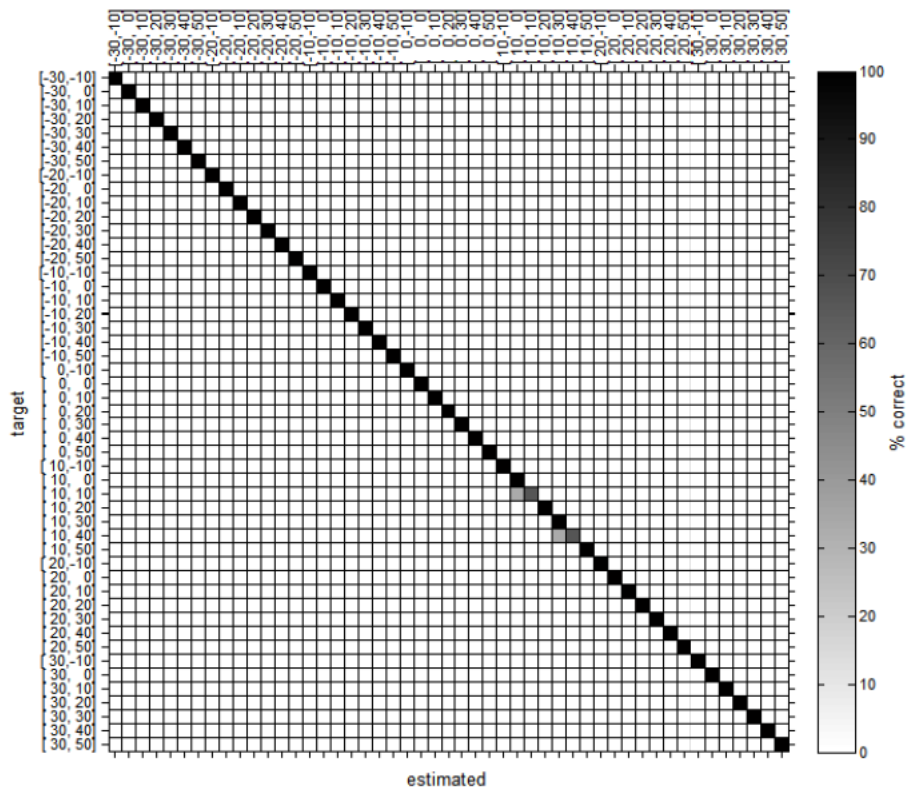
(a) Rauschen; Vortraining mit Bottleneck; frontal; parallel; ILD-ITD



(b) Rauschen; Vortraining mit Bottleneck; frontal; parallel; ILD

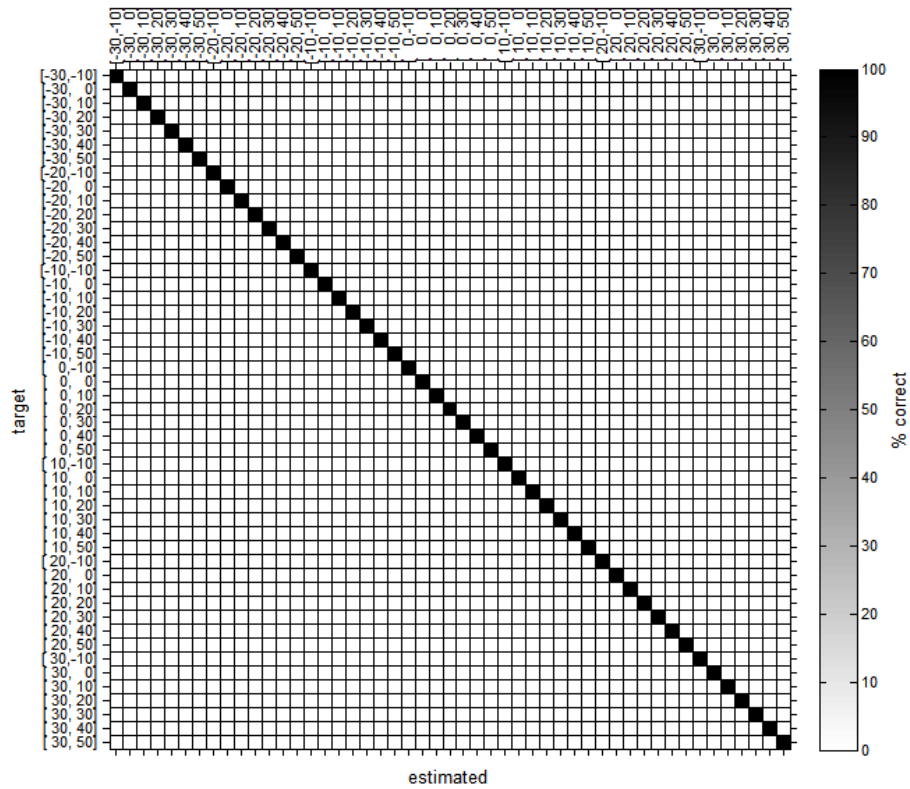


(c) Rauschen; Vortraining mit Bottleneck; frontal; parallel; ITD



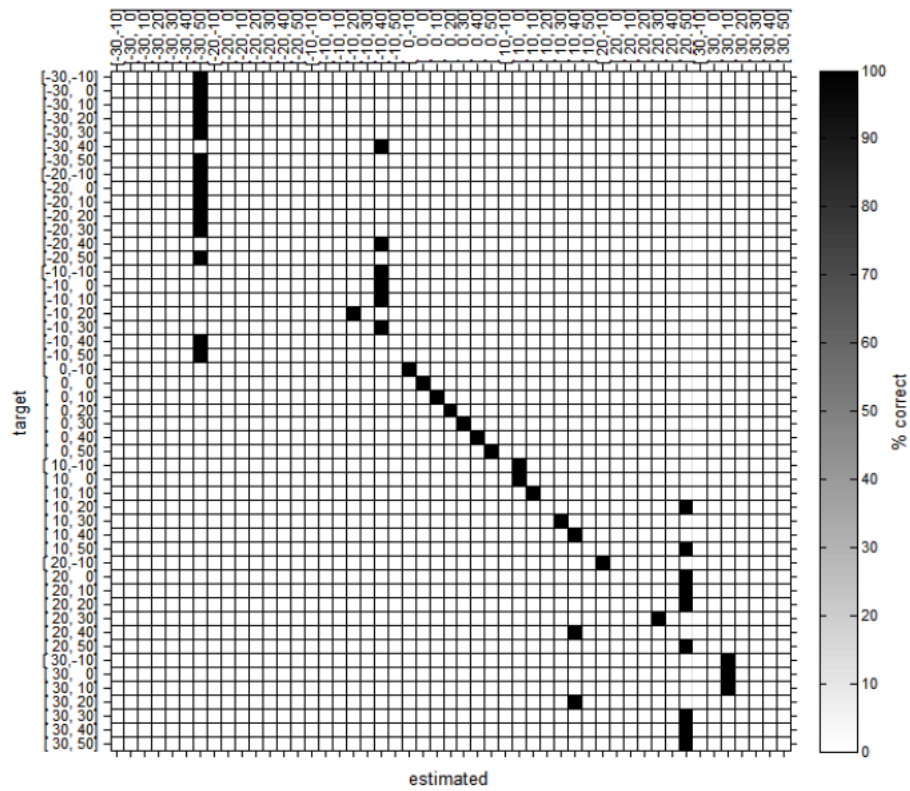
(d) Rauschen; Vortraining mit Bottleneck; frontal; parallel; Betragsspektrum



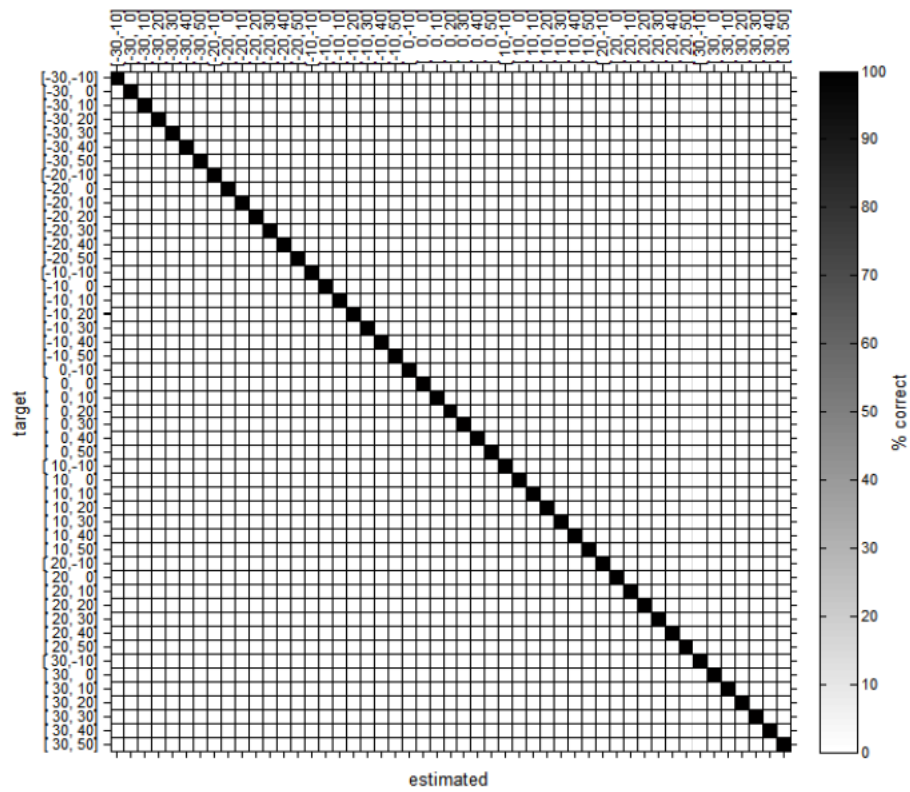


(e) Rauschen; Vortraining mit Bottleneck; frontal; parallel; **mfcc**

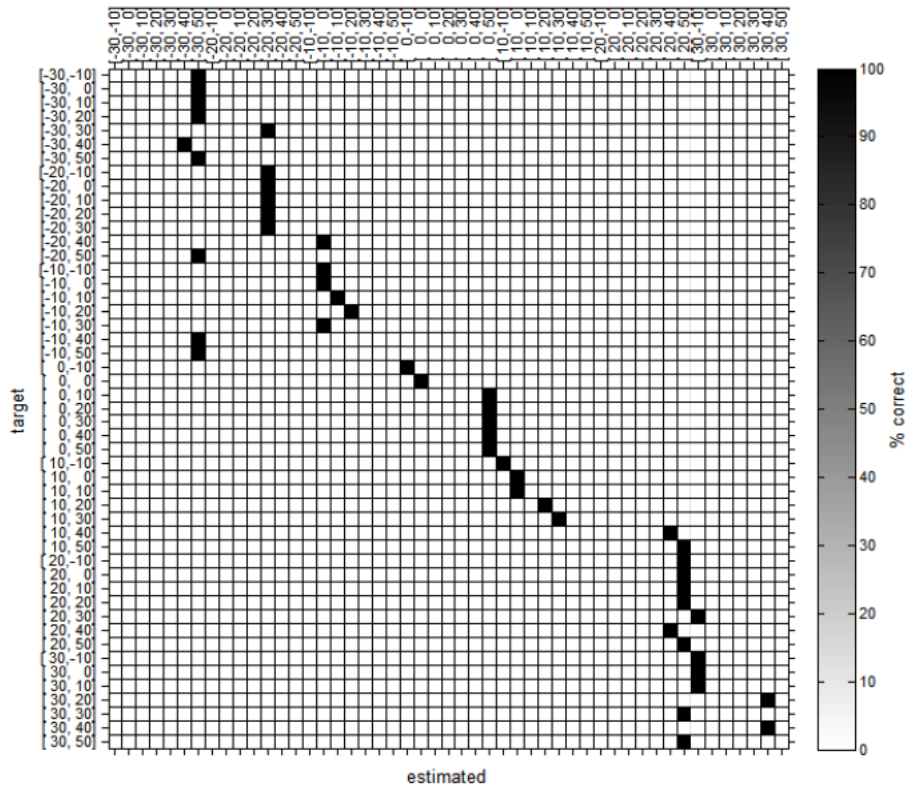
Abbildung A.1: Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; frontal; parallel;



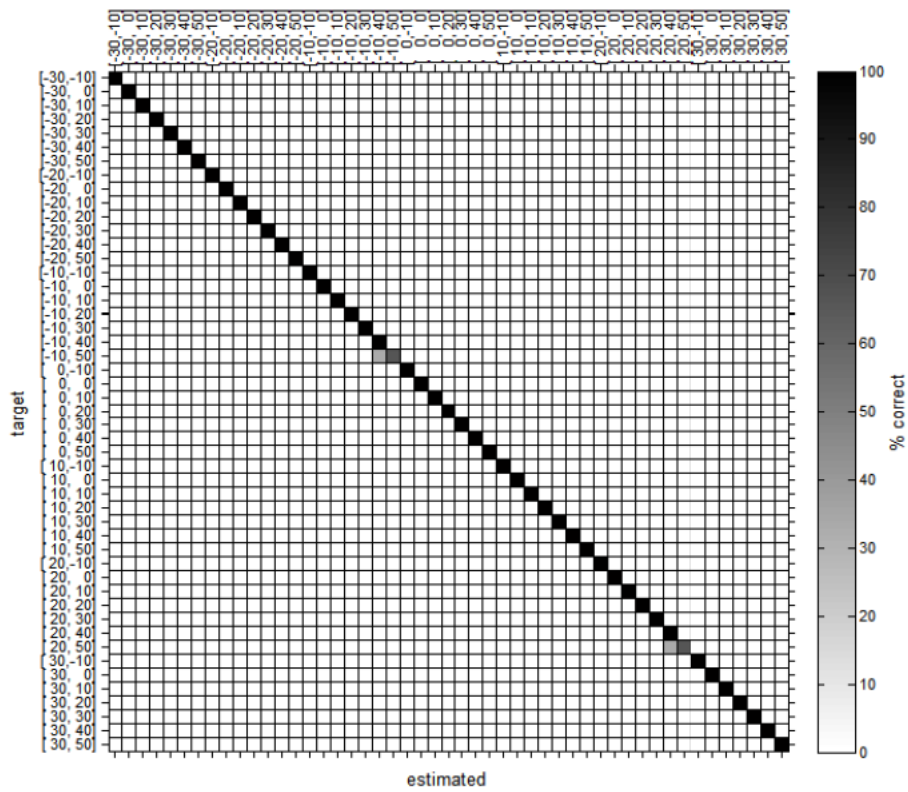
(a) Rauschen; Vortraining mit Bottleneck; frontal; angehängt; ILD-ITD



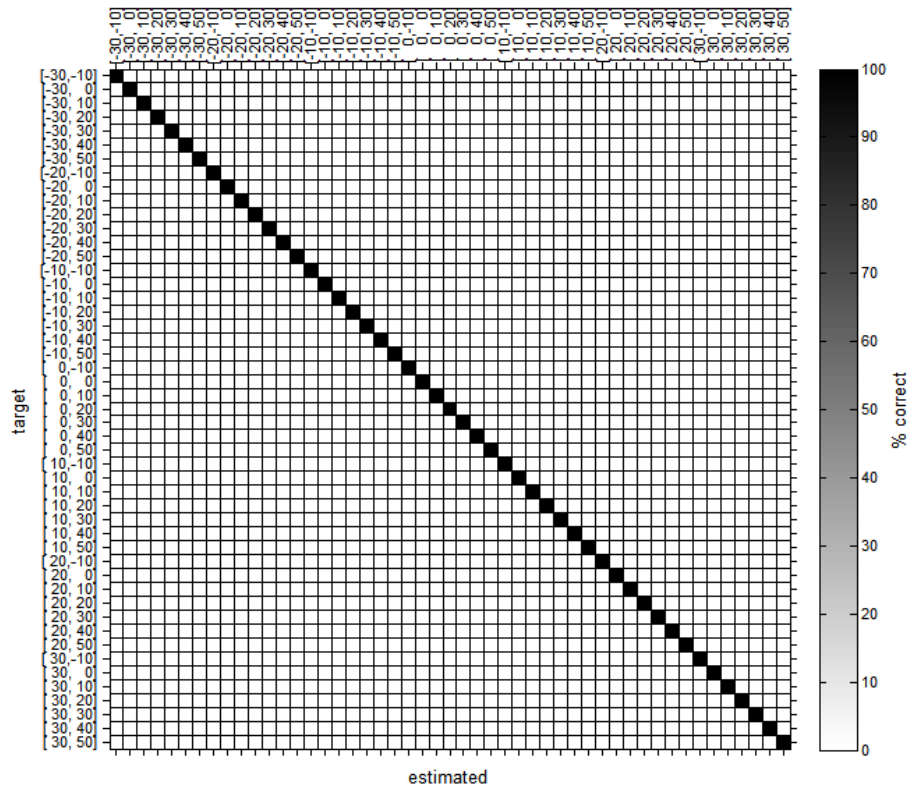
(b) Rauschen; Vortraining mit Bottleneck; frontal; angehängt; ILD



(c) Rauschen; Vortraining mit Bottleneck; frontal; angehängt; ITD

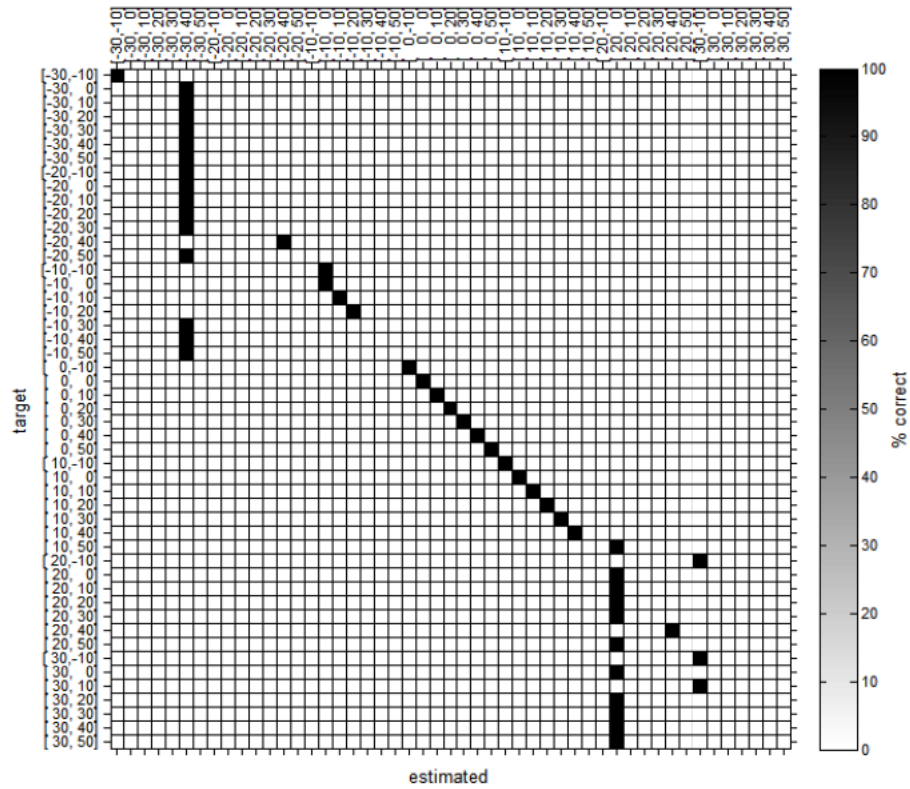


(d) Rauschen; Vortraining mit Bottleneck; frontal; angehängt; Betragsspektrum

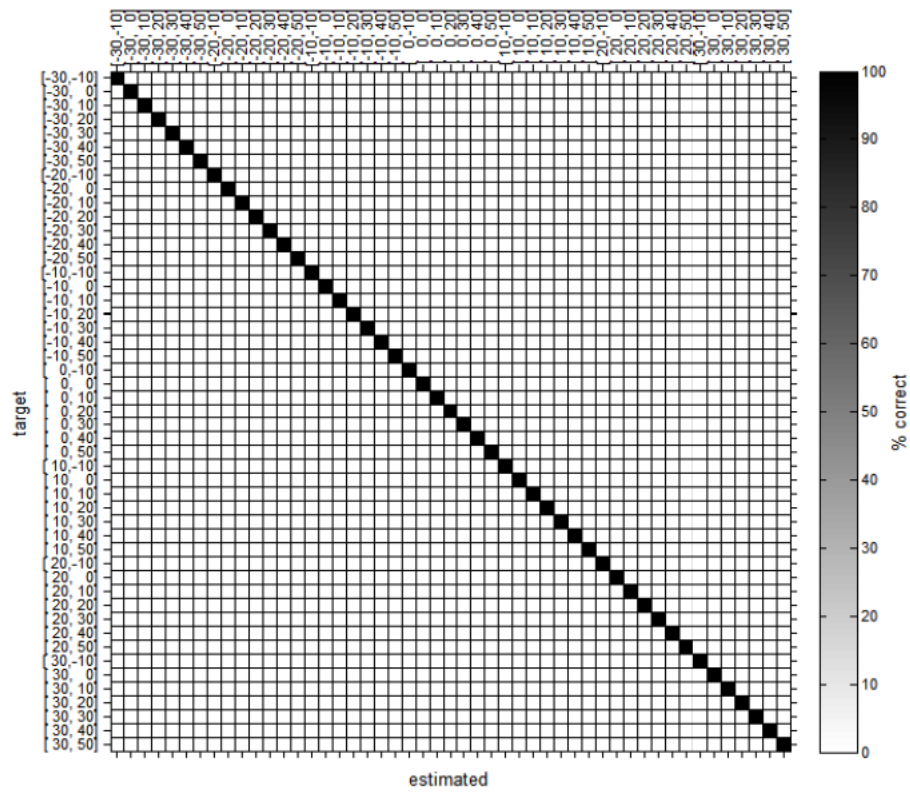


(e) Rauschen; Vortraining mit Bottleneck; frontal; angehängt; **mfcc**

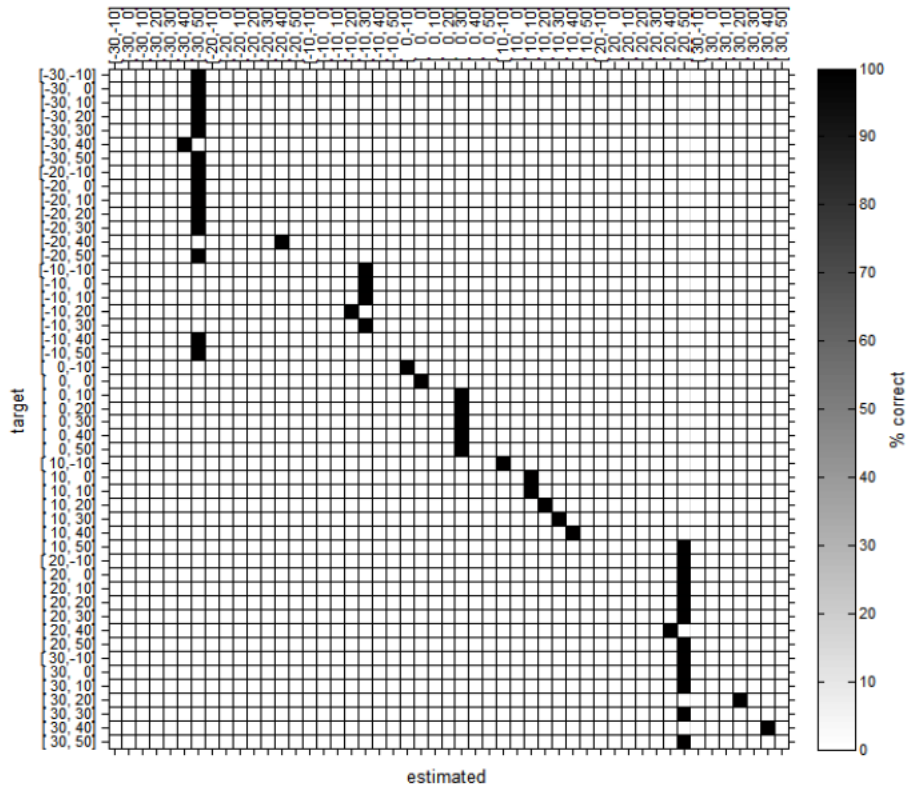
Abbildung A.2: Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; frontal; angehängt;



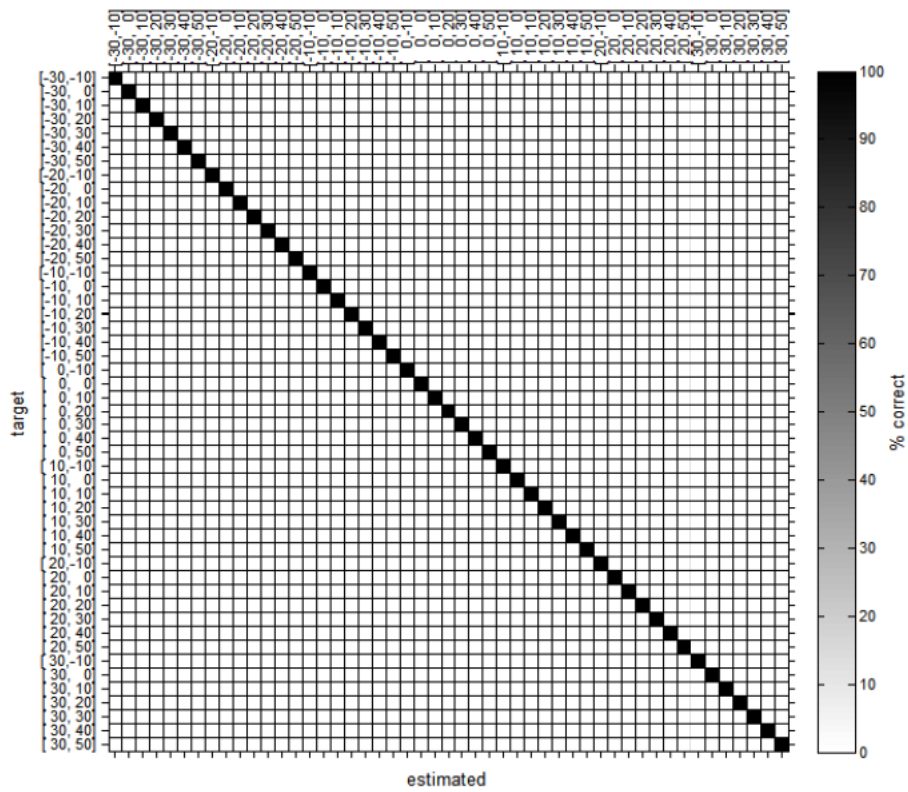
(a) Rauschen; Vortraining mit Bottleneck; frontal; sphärisch; ILD-ITD



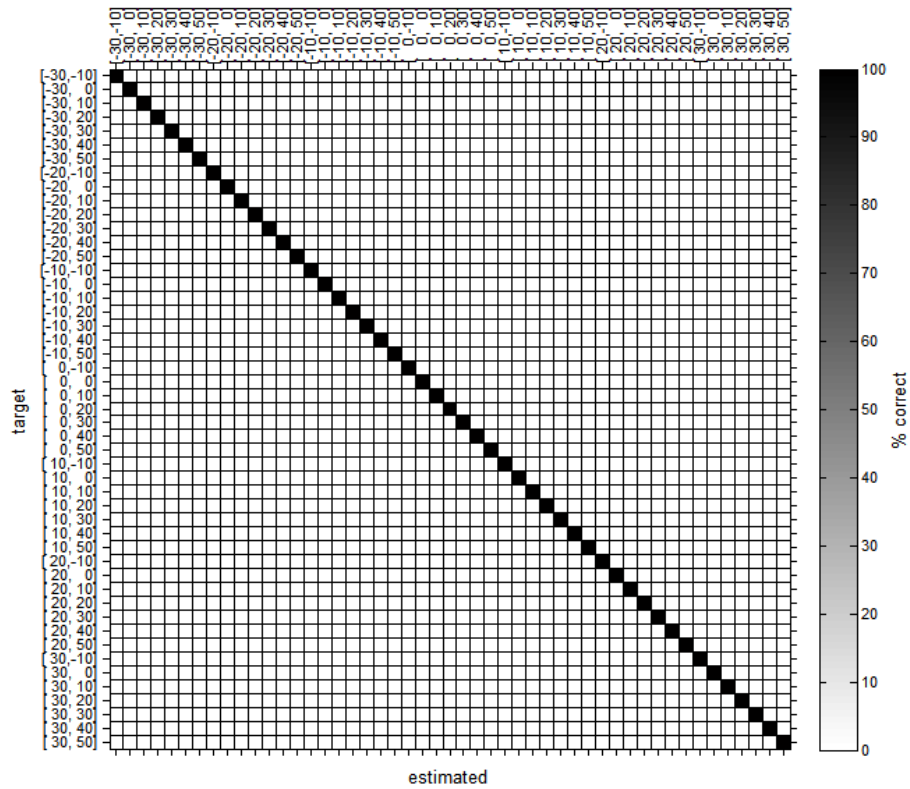
(b) Rauschen; Vortraining mit Bottleneck; frontal; sphärisch; ILD



(c) Rauschen; Vortraining mit Bottleneck; frontal; sphärisch; ITD



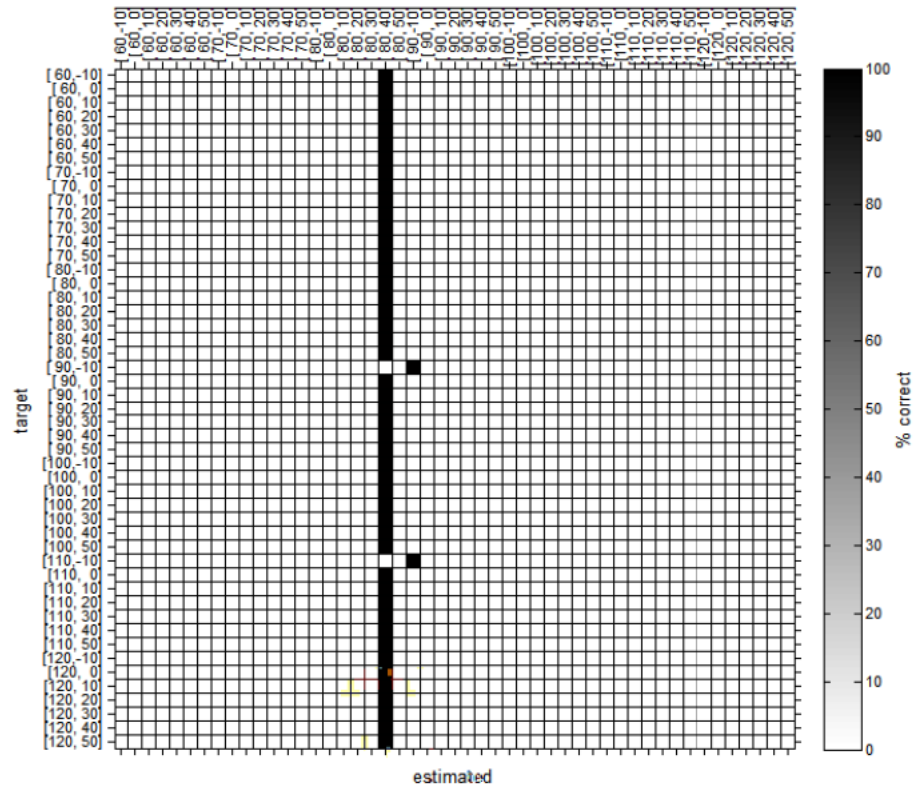
(d) Rauschen; Vortraining mit Bottleneck; frontal; sphärisch; Betragsspektrum



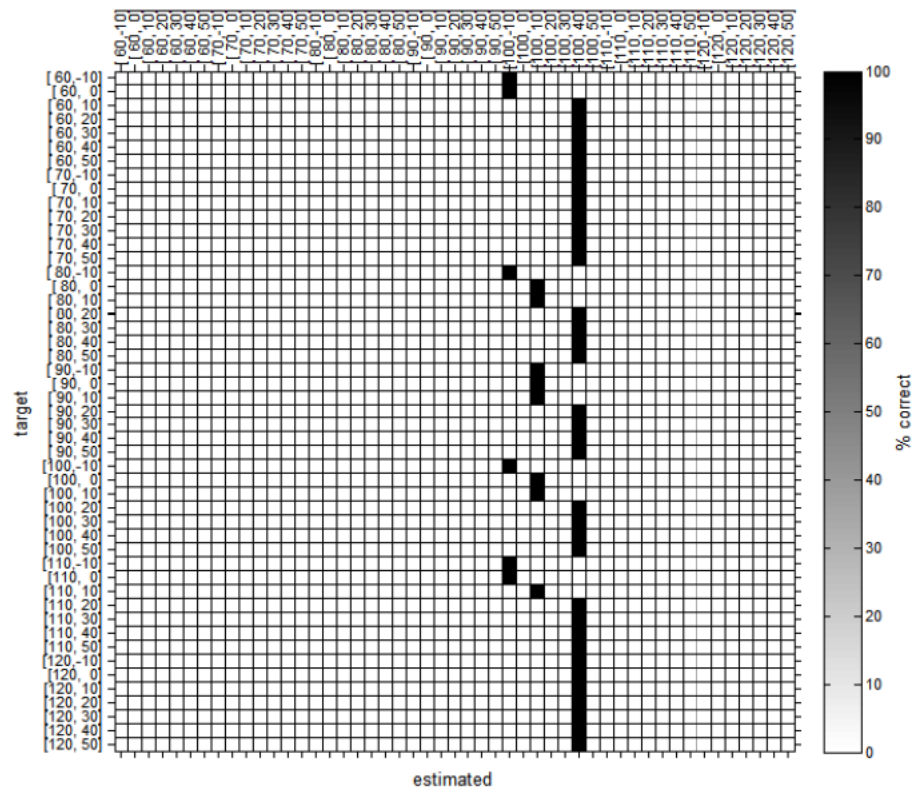
(e) Rauschen; Vortraining mit Bottleneck; frontal; sphärisch; **mfcc**

Abbildung A.3: Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; frontal; sphärisch;



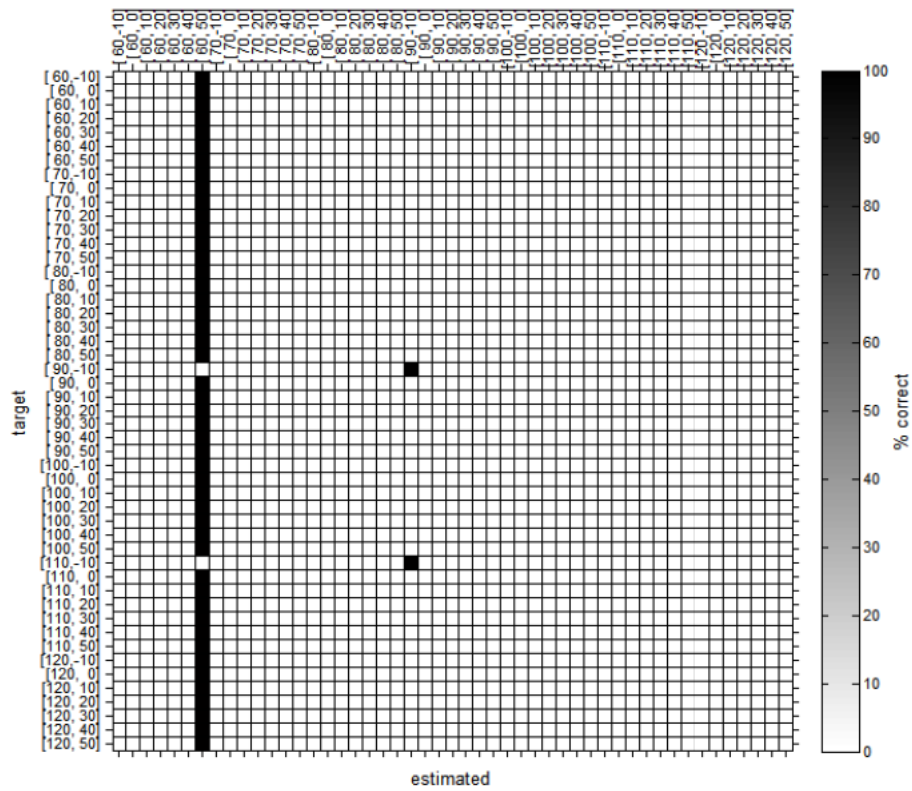


(a) Rauschen; Vortraining mit Bottleneck; seitlich; parallel; ILD-ITD

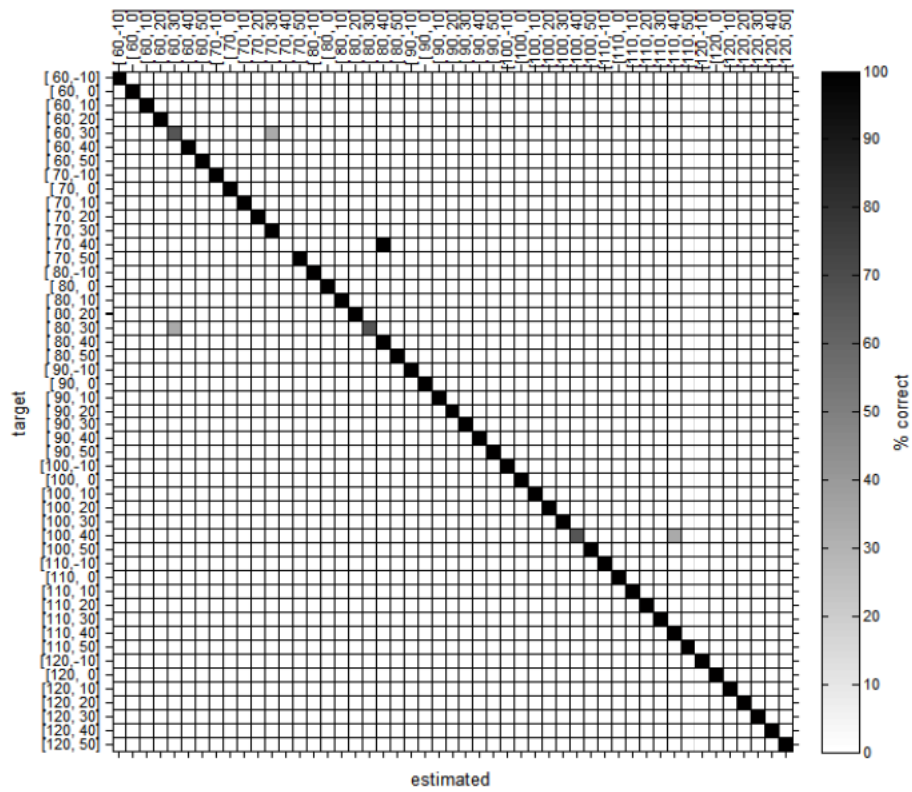


(b) Rauschen; Vortraining mit Bottleneck; seitlich; parallel; ILD

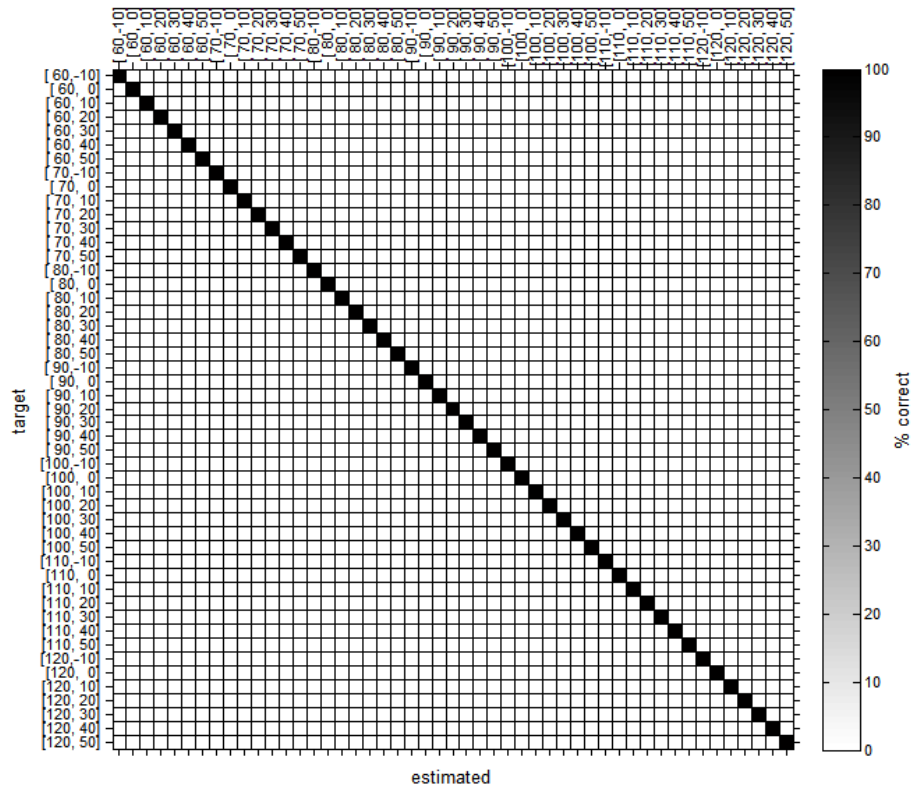




(c) Rauschen; Vortraining mit Bottleneck; seitlich; parallel; ITD

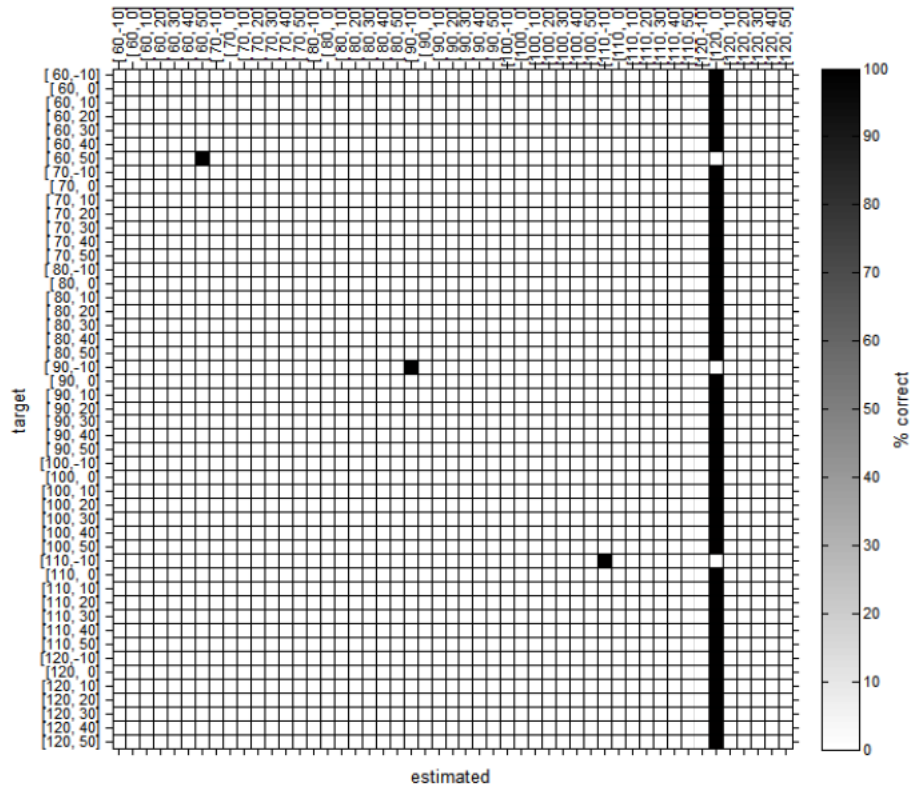


(d) Rauschen; Vortraining mit Bottleneck; seitlich; parallel; Betragsspektrum

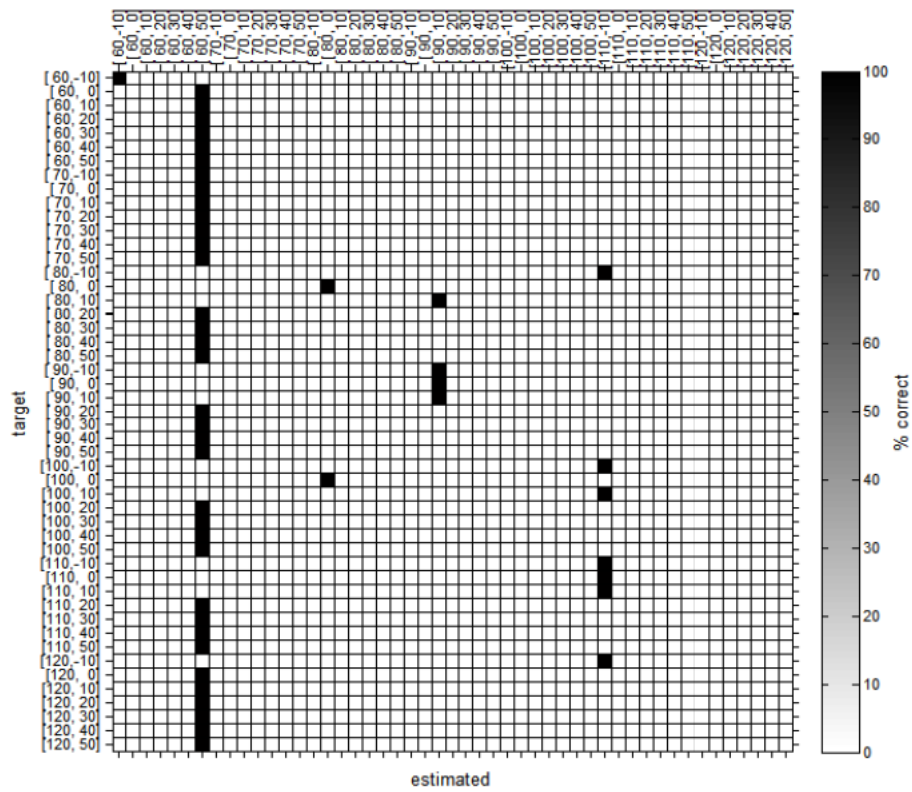


(e) Rauschen; Vortraining mit Bottleneck; seitlich; parallel; **mfcc**

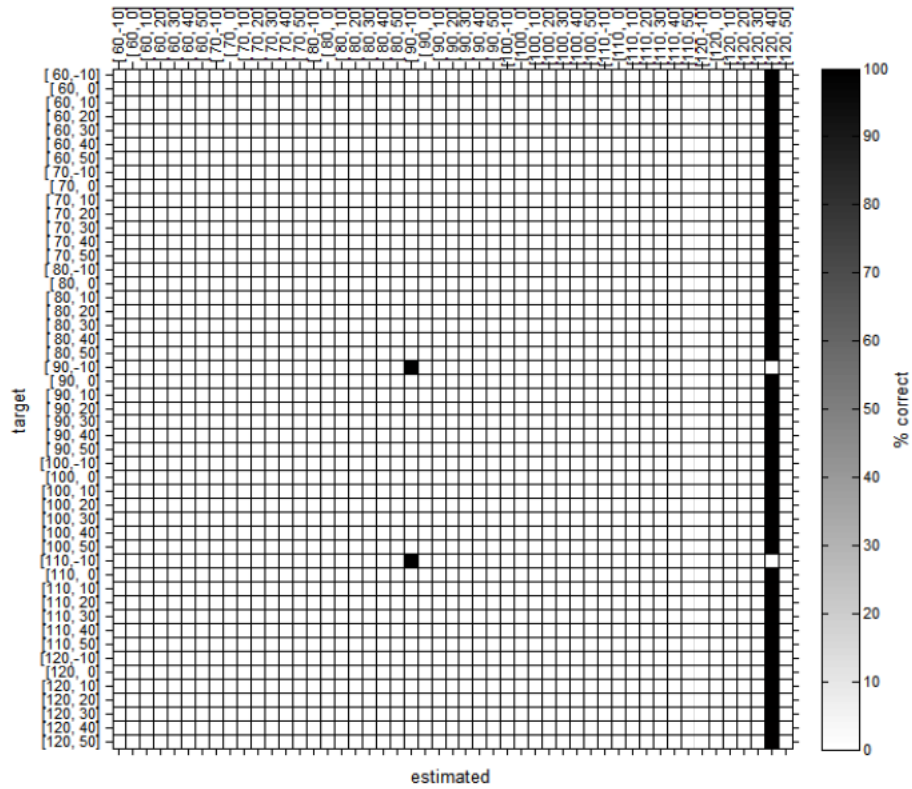
Abbildung A.4: Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; seitlich; parallel;



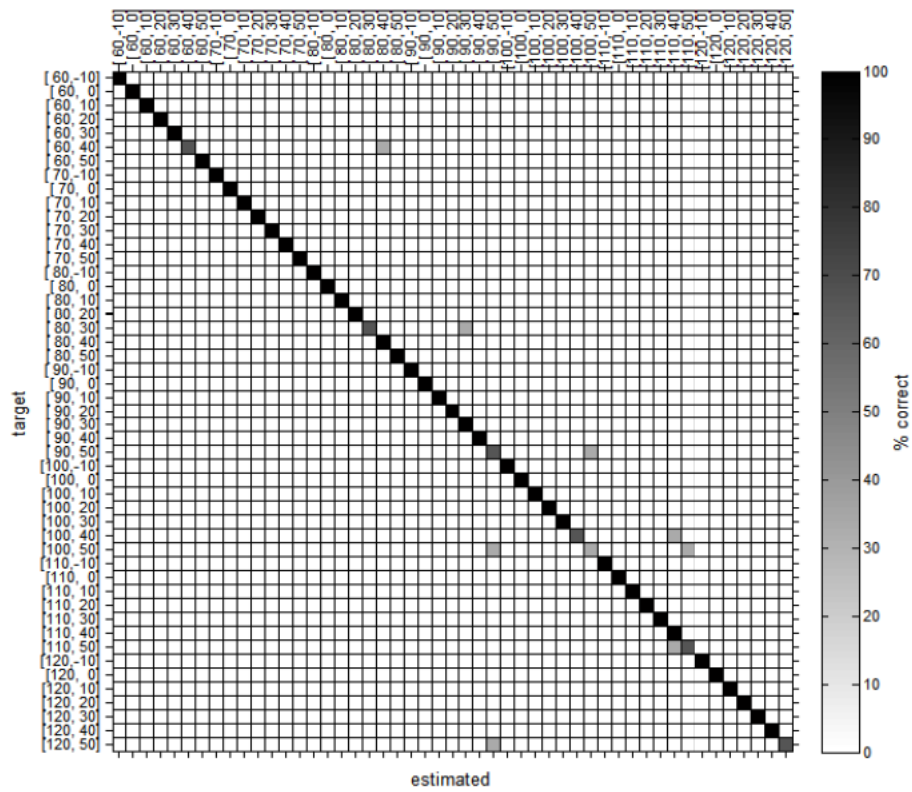
(a) Rauschen; Vortraining mit Bottleneck; seitlich; angehängt; ILD-ITD



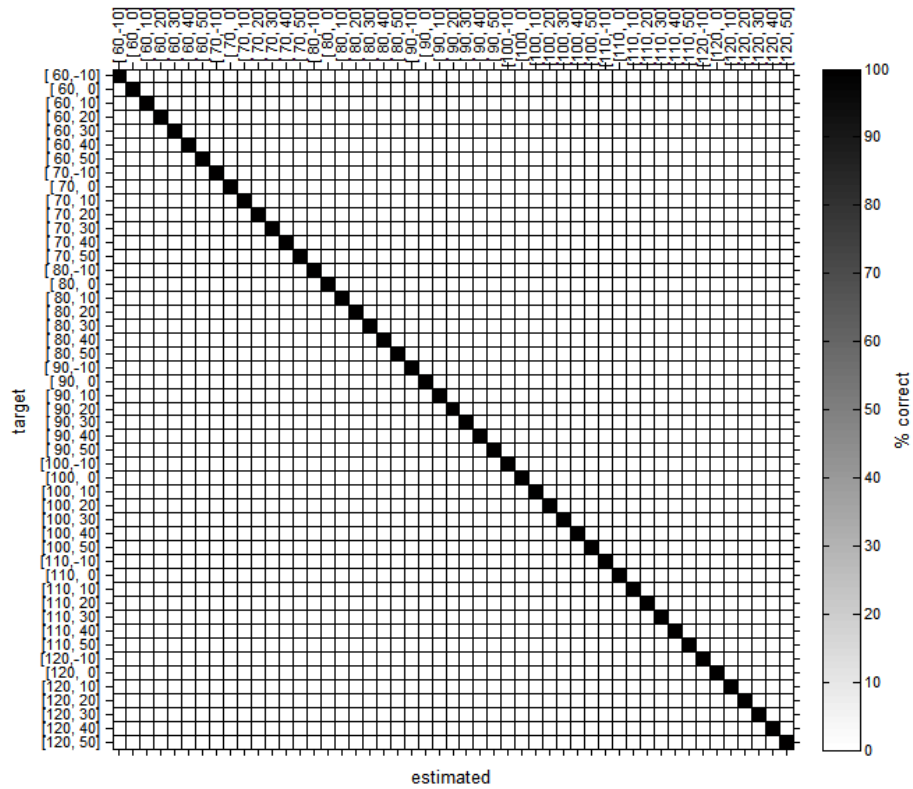
(b) Rauschen; Vortraining mit Bottleneck; seitlich; angehängt; ILD



(c) Rauschen; Vortraining mit Bottleneck; seitlich; angehängt; ITD

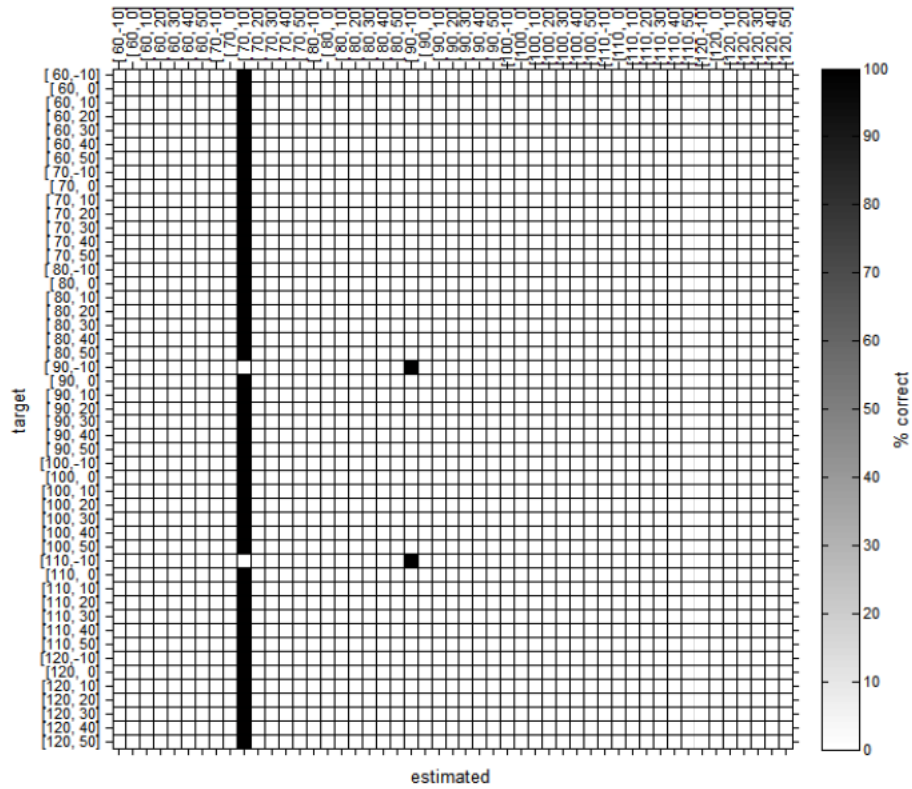


(d) Rauschen; Vortraining mit Bottleneck; seitlich; angehängt; Betragsspektrum

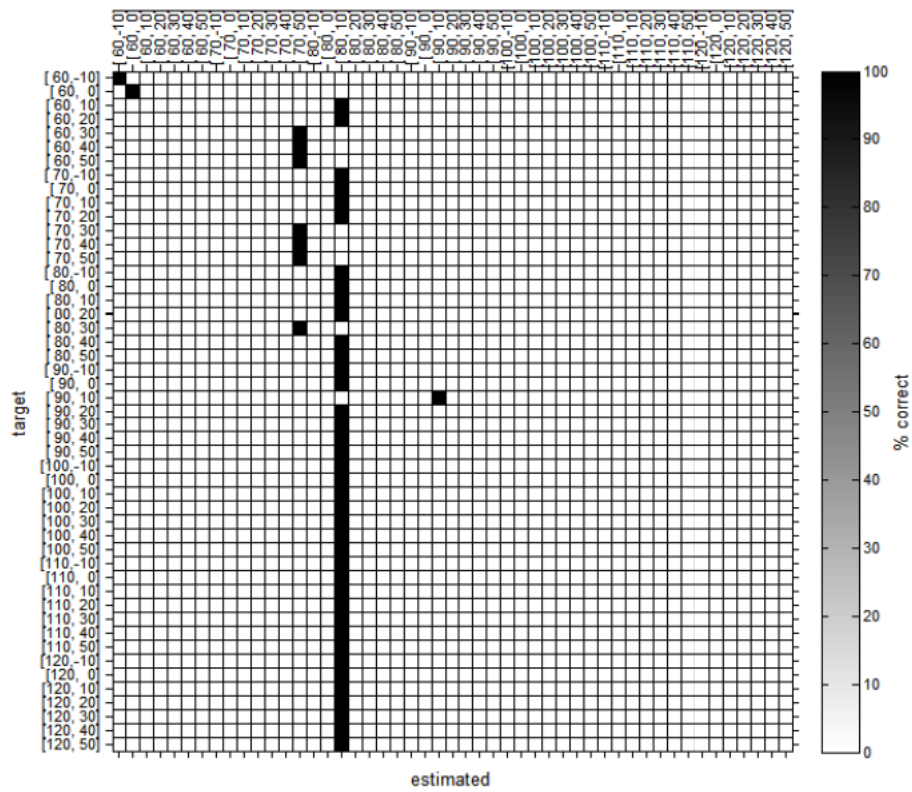


(e) Rauschen; Vortraining mit Bottleneck; seitlich; angehängt; **mfcc**

Abbildung A.5: Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; seitlich; angehängt;

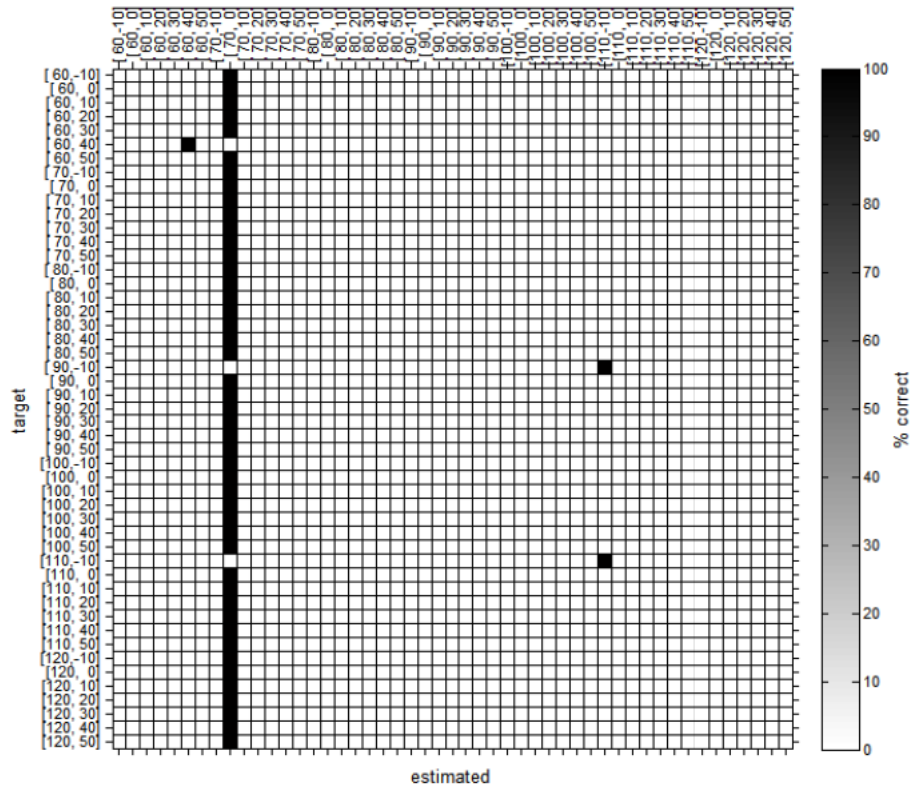


(a) Rauschen; Vortraining mit Bottleneck; seitlich; sphärisch; ILD-ITD

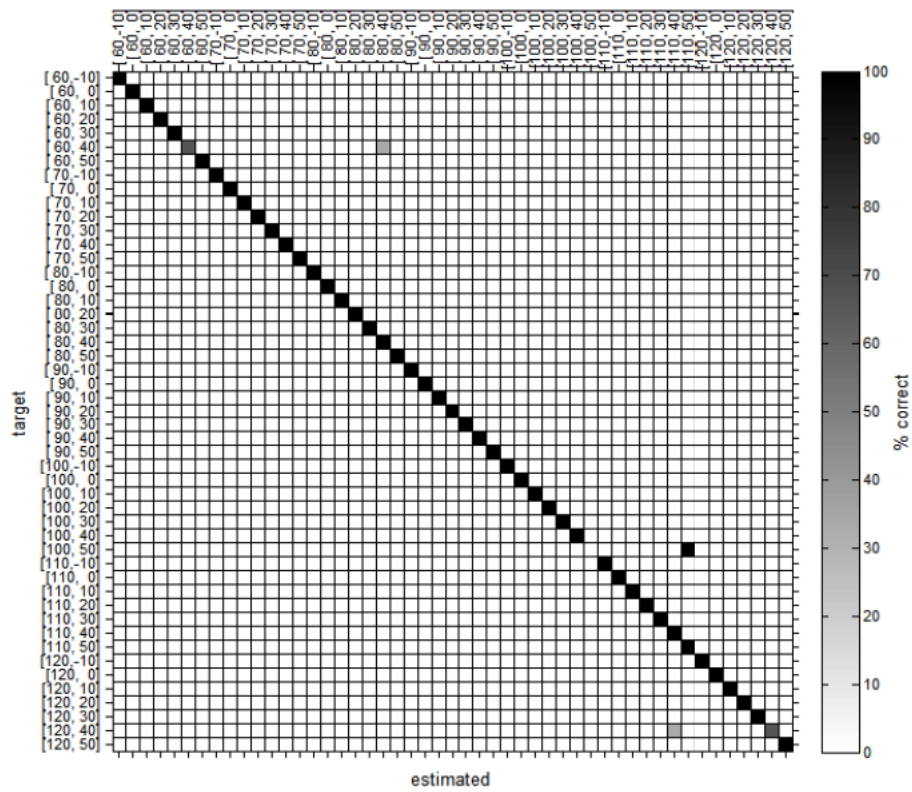


(b) Rauschen; Vortraining mit Bottleneck; seitlich; sphärisch; ILD

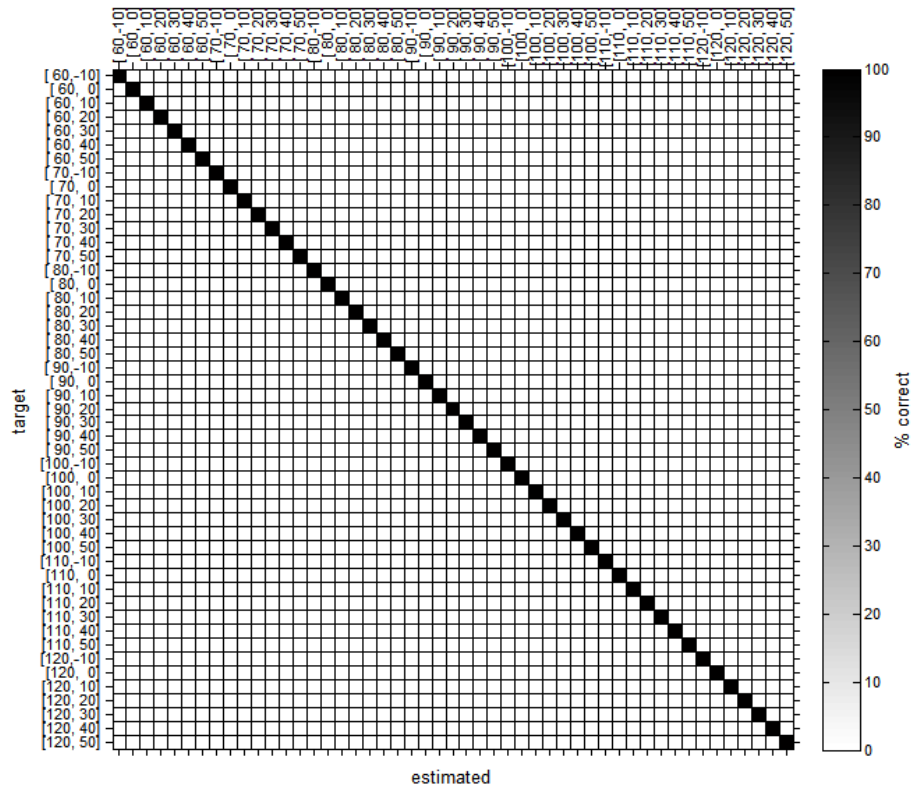




(c) Rauschen; Vortraining mit Bottleneck; seitlich; sphärisch; ITD



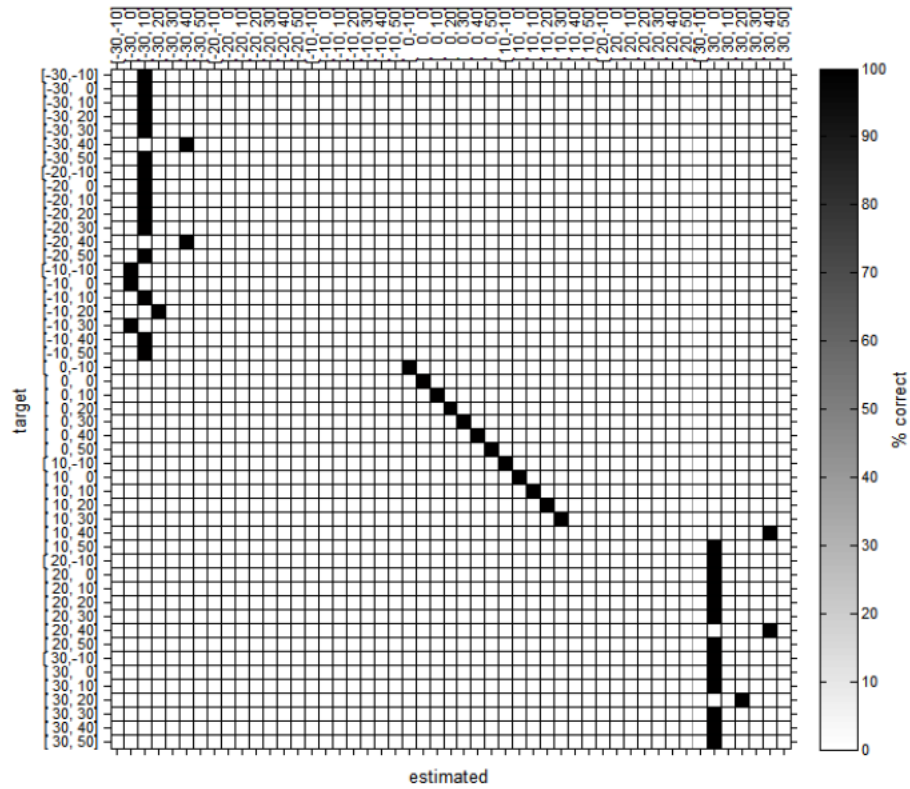
(d) Rauschen; Vortraining mit Bottleneck; seitlich; sphärisch; Betragsspektrum



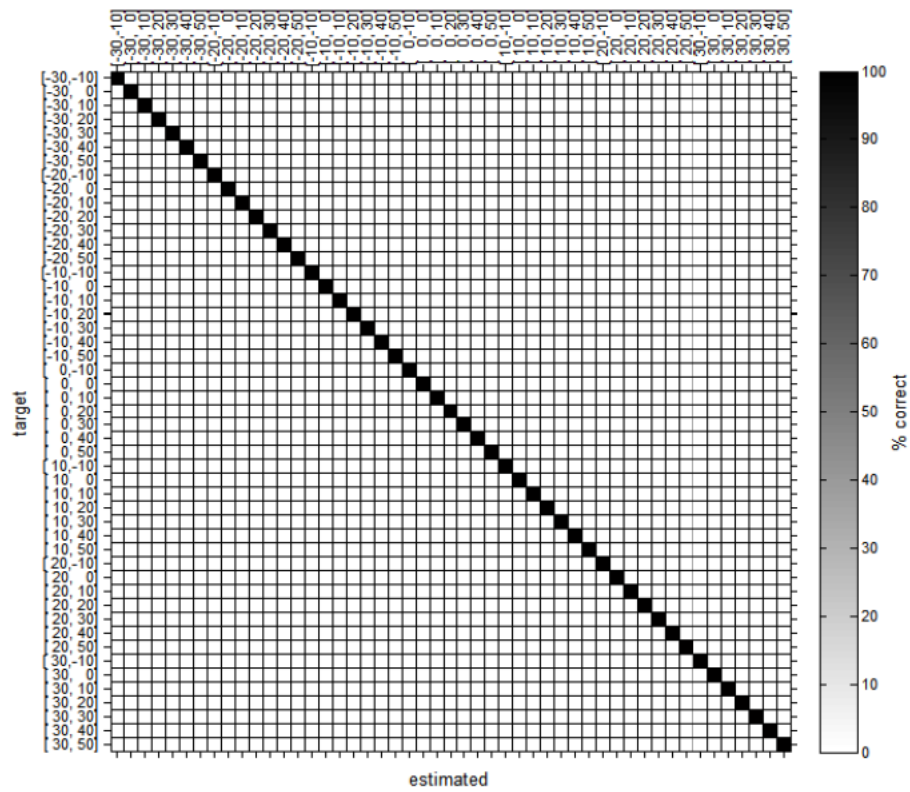
(e) Rauschen; Vortraining mit Bottleneck; seitlich; sphärisch; **mfcc**

Abbildung A.6: Confusion-Matrizen für die Konditionen: Rauschen; Vortraining mit Bottleneck; seitlich; sphärisch;

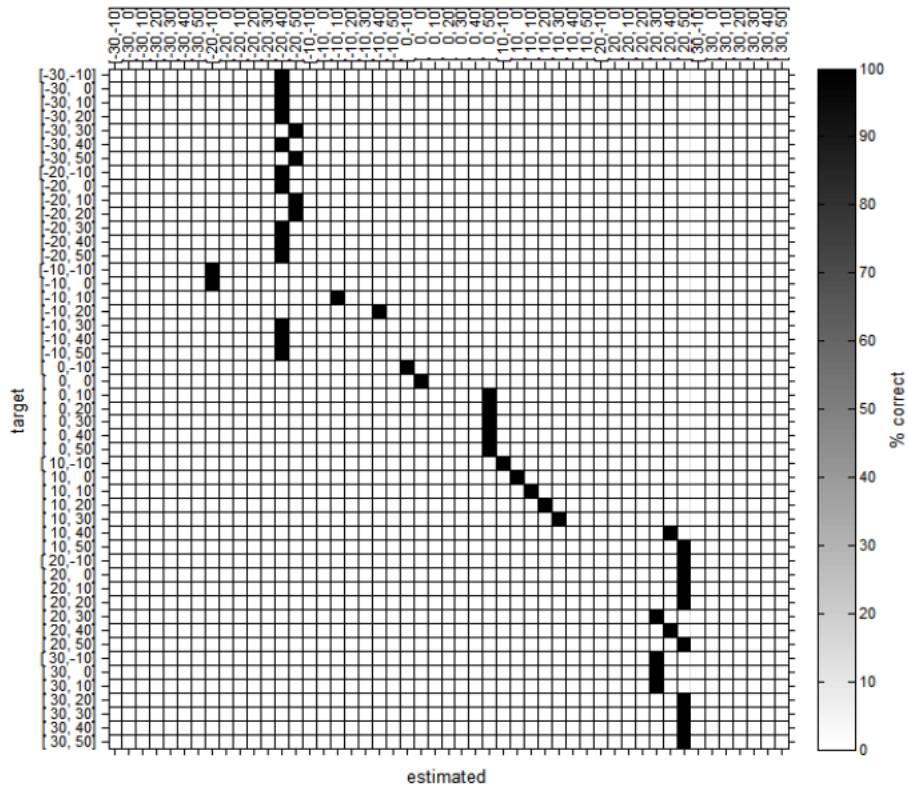




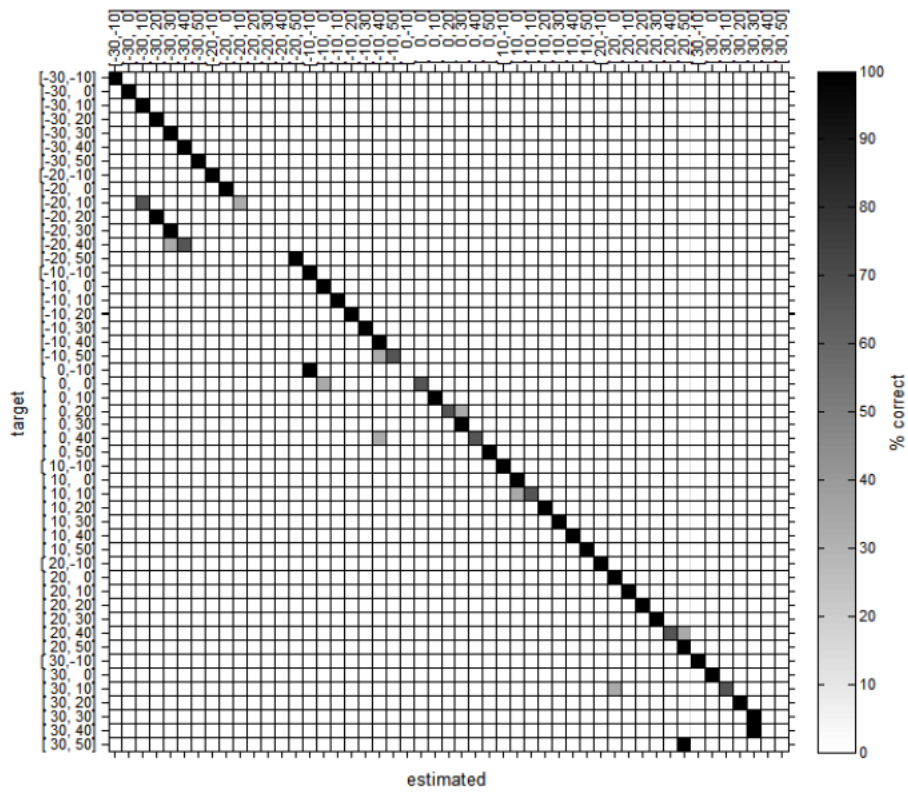
(a) Rauschen; ohne Bottleneck-Vortraining; frontal; parallel; ILD-ITD



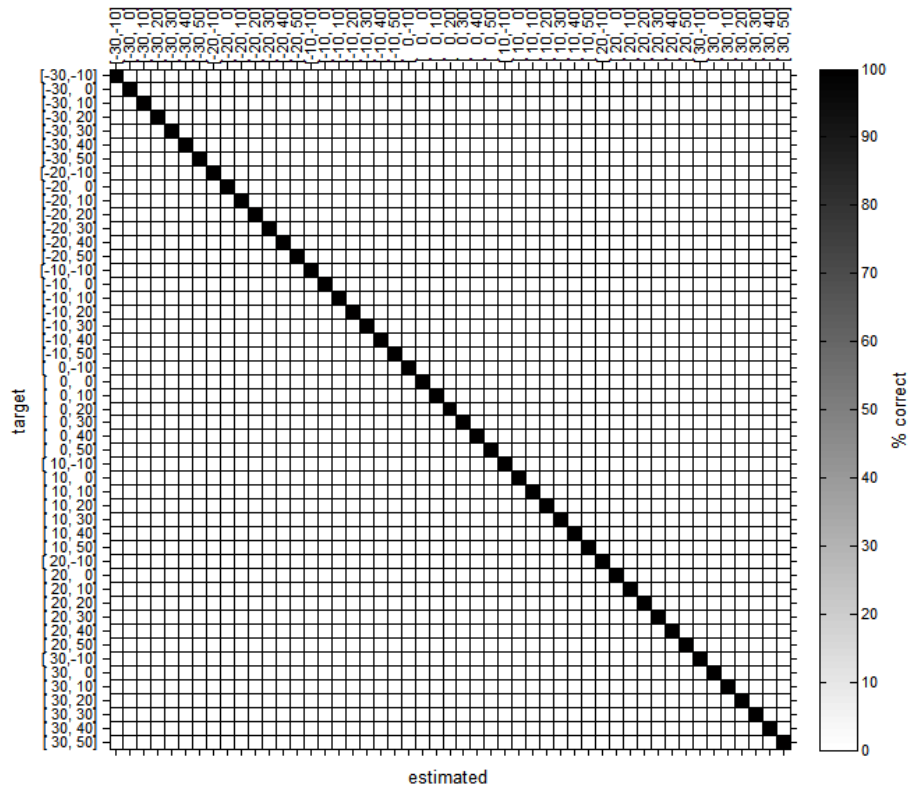
(b) Rauschen; ohne Bottleneck-Vortraining; frontal; parallel; ILD



(c) Rauschen; ohne Bottleneck-Vortraining; frontal; parallel; ITD

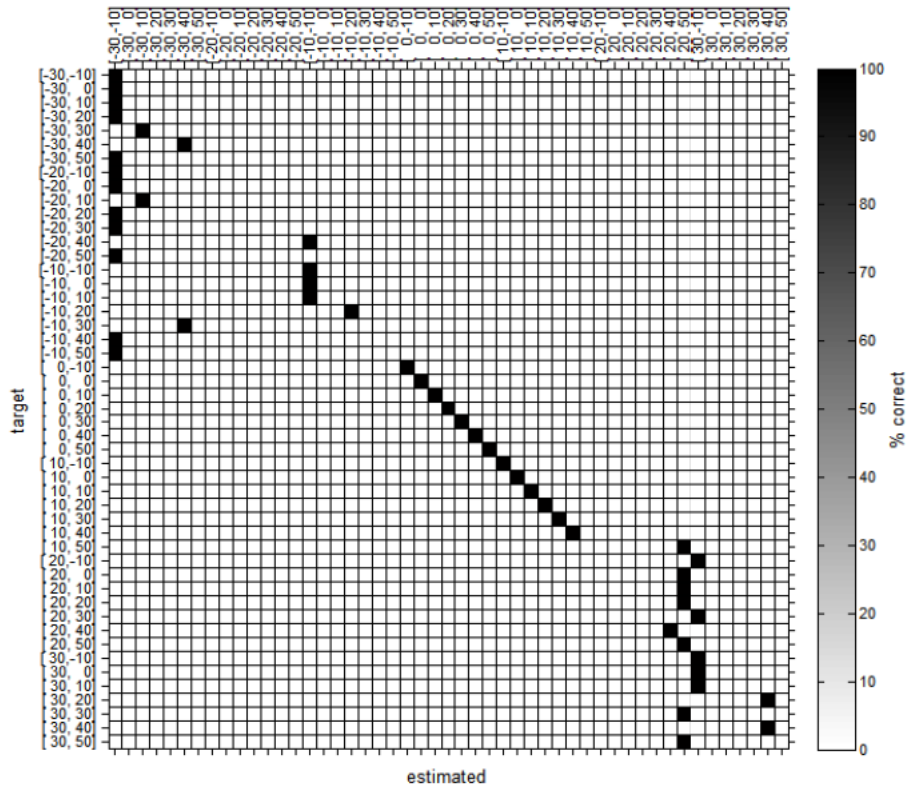


(d) Rauschen; ohne Bottleneck-Vortraining; frontal; parallel; Betragsspektrum

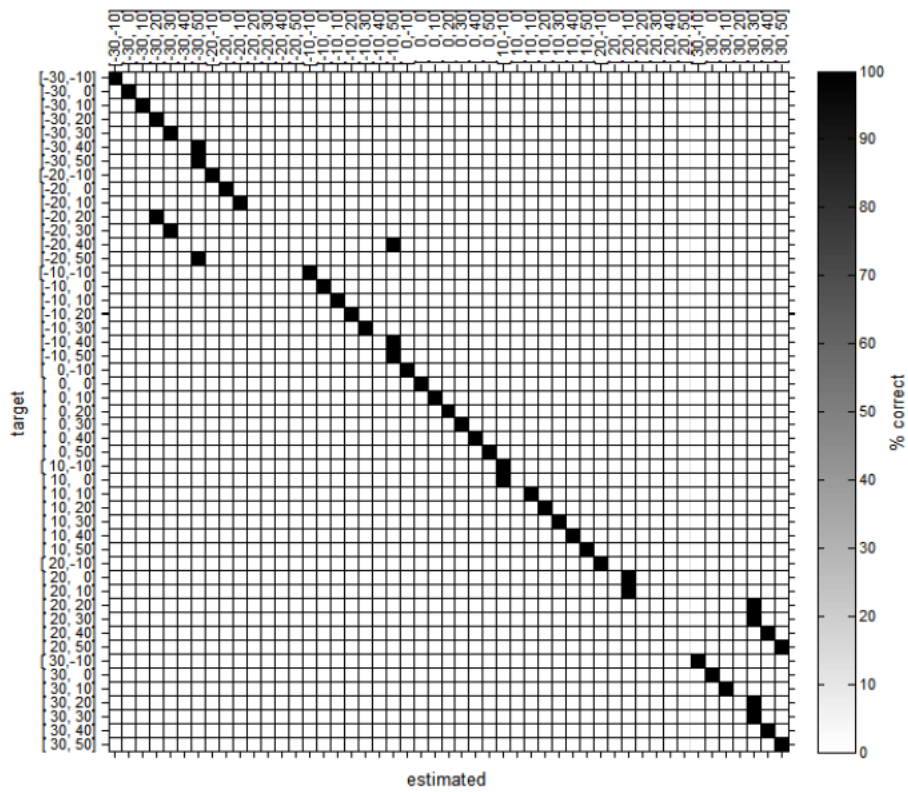


(e) Rauschen; ohne Bottleneck-Vortraining; frontal; parallel; **mfcc**

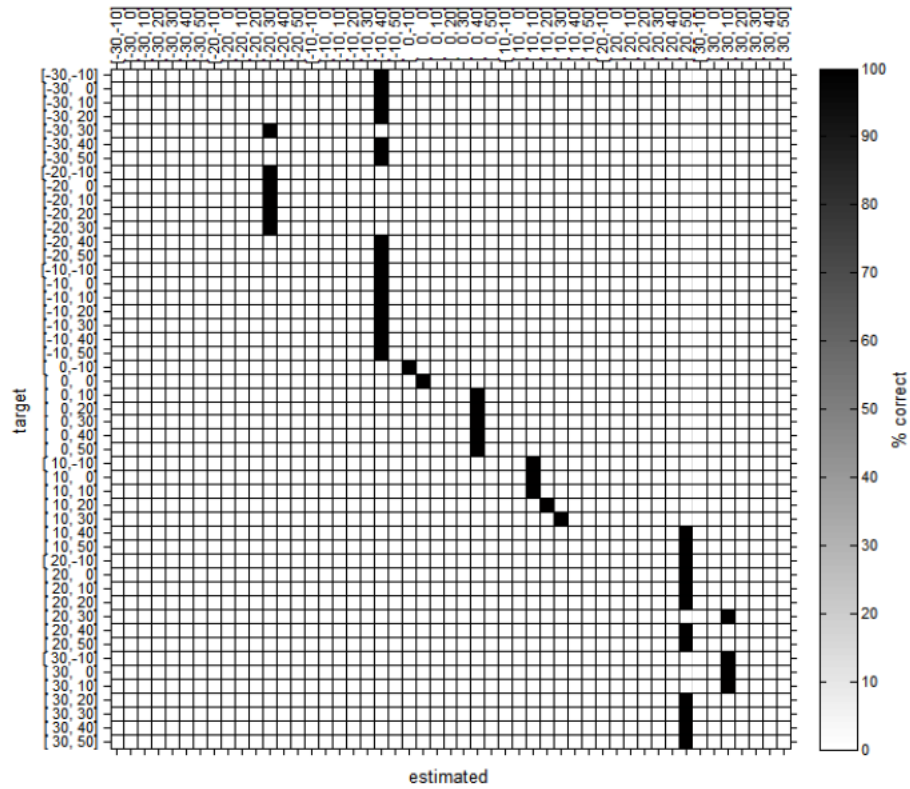
Abbildung A.7: Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; frontal; parallel;



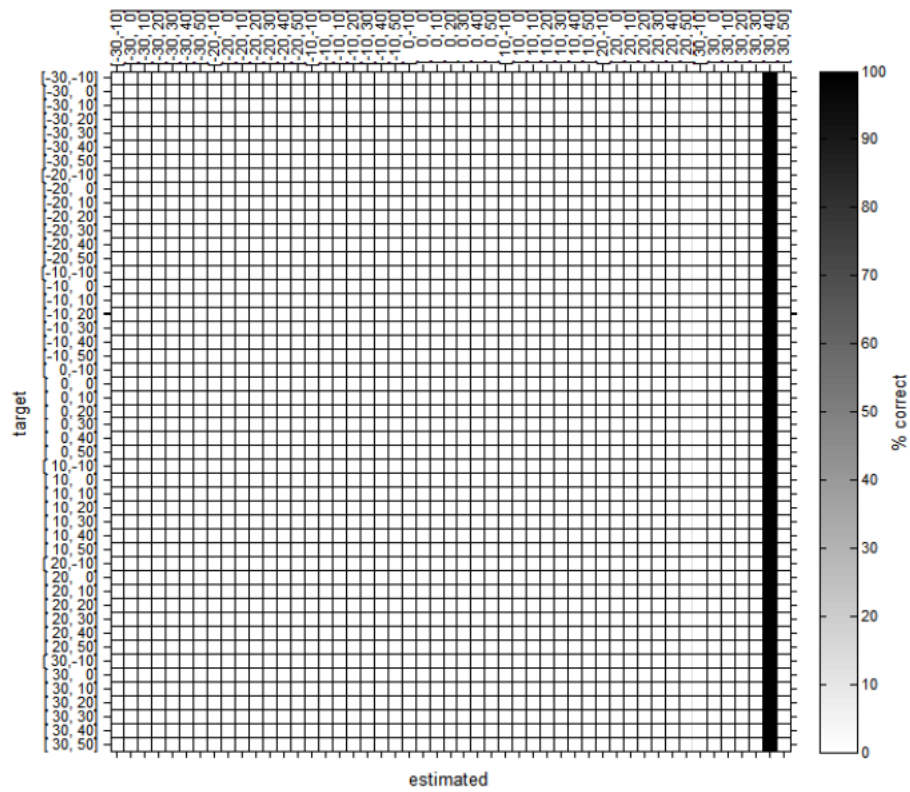
(a) Rauschen; ohne Bottleneck-Vortraining; frontal; angehängt; ILD-ITD



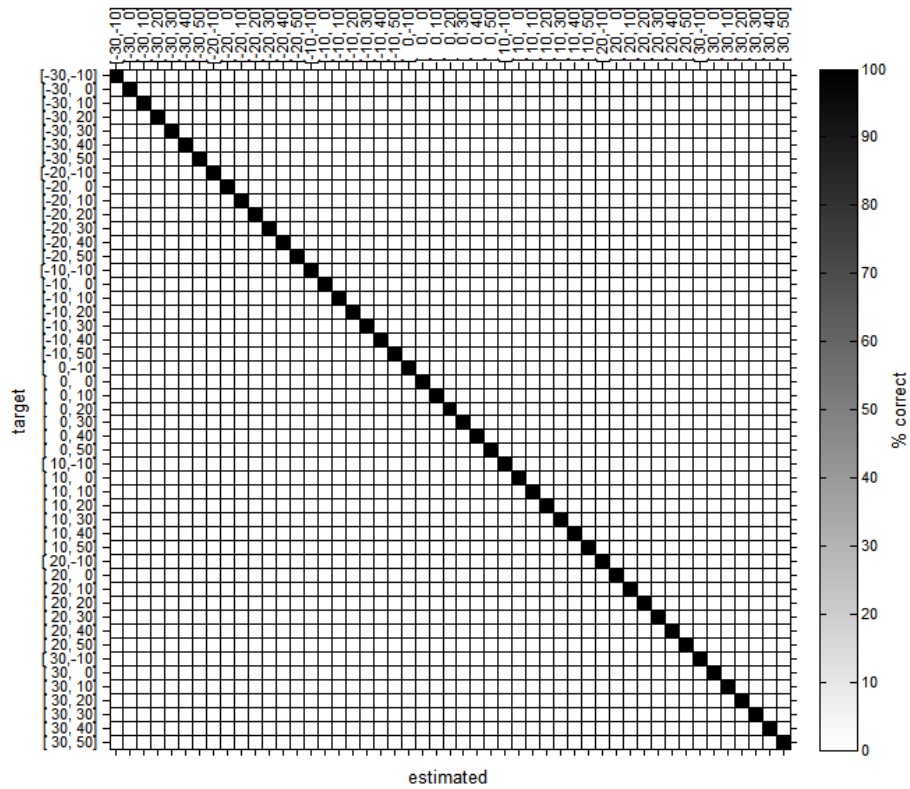
(b) Rauschen; ohne Bottleneck-Vortraining; frontal; angehängt; ILD



(c) Rauschen; ohne Bottleneck-Vortraining; frontal; angehängt; ITD



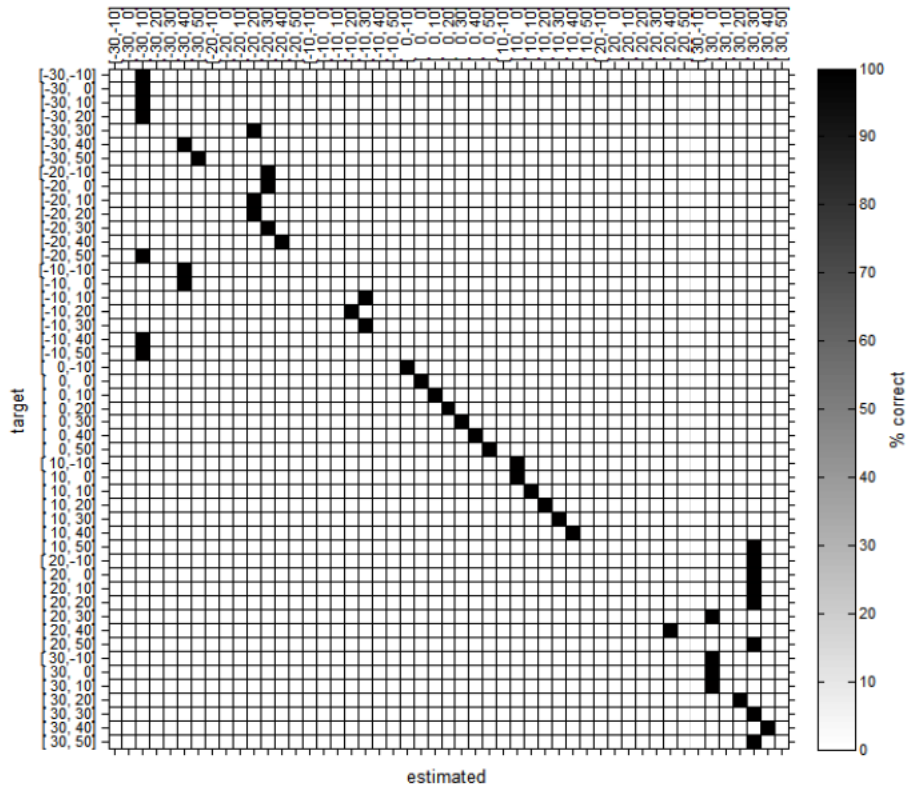
(d) Rauschen; ohne Bottleneck-Vortraining; frontal; angehängt; Betragsspektrum



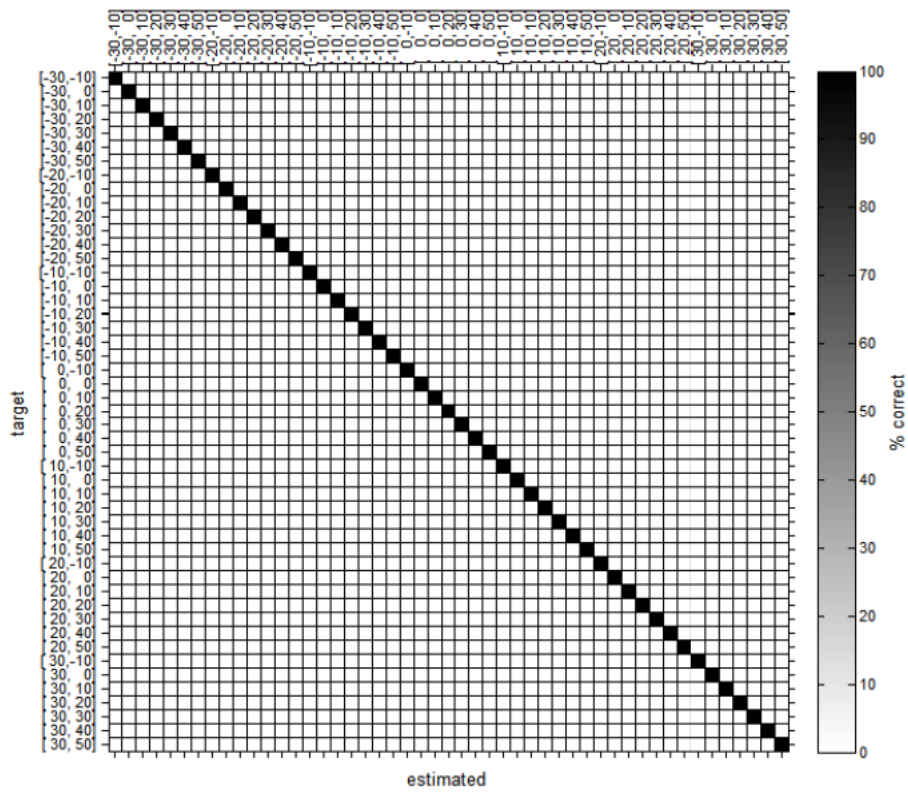
(e) Rauschen; ohne Bottleneck-Vortraining; frontal; angehängt; **mfcc**

Abbildung A.8: Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; frontal; angehängt;

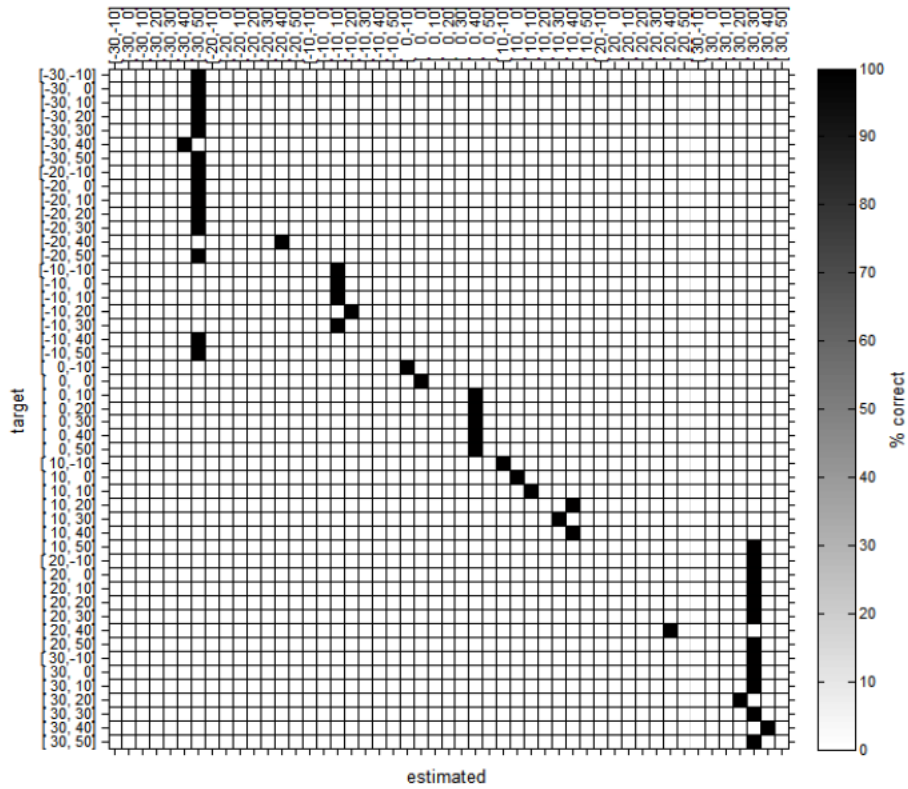




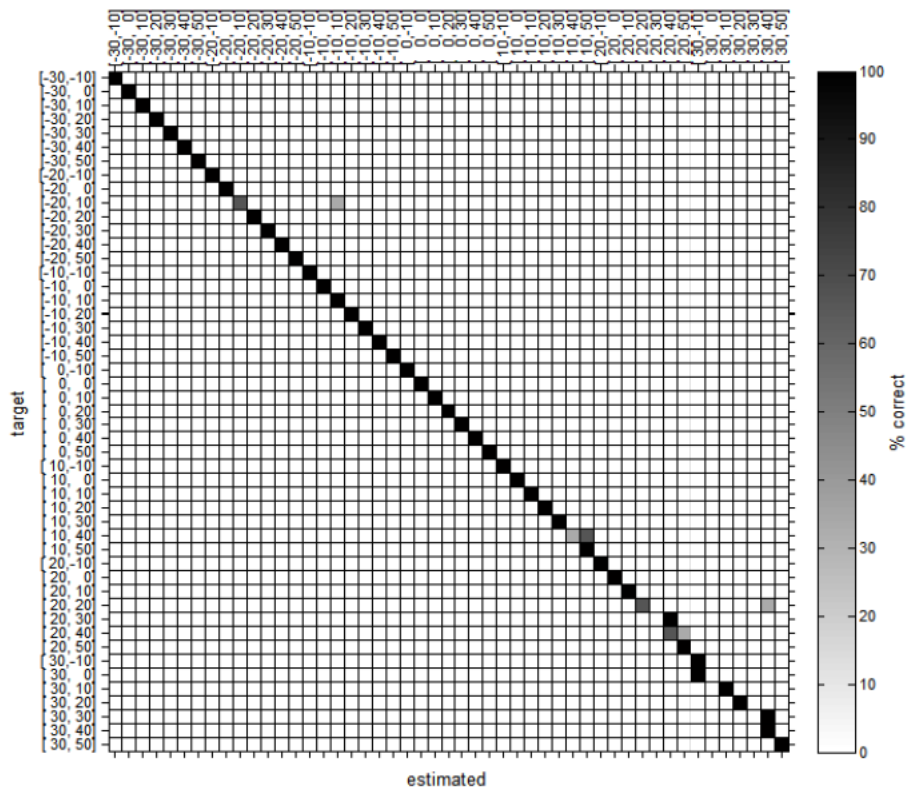
(a) Rauschen; ohne Bottleneck-Vortraining; frontal; sphärisch; ILD-ITD



(b) Rauschen; ohne Bottleneck-Vortraining; frontal; sphärisch; ILD

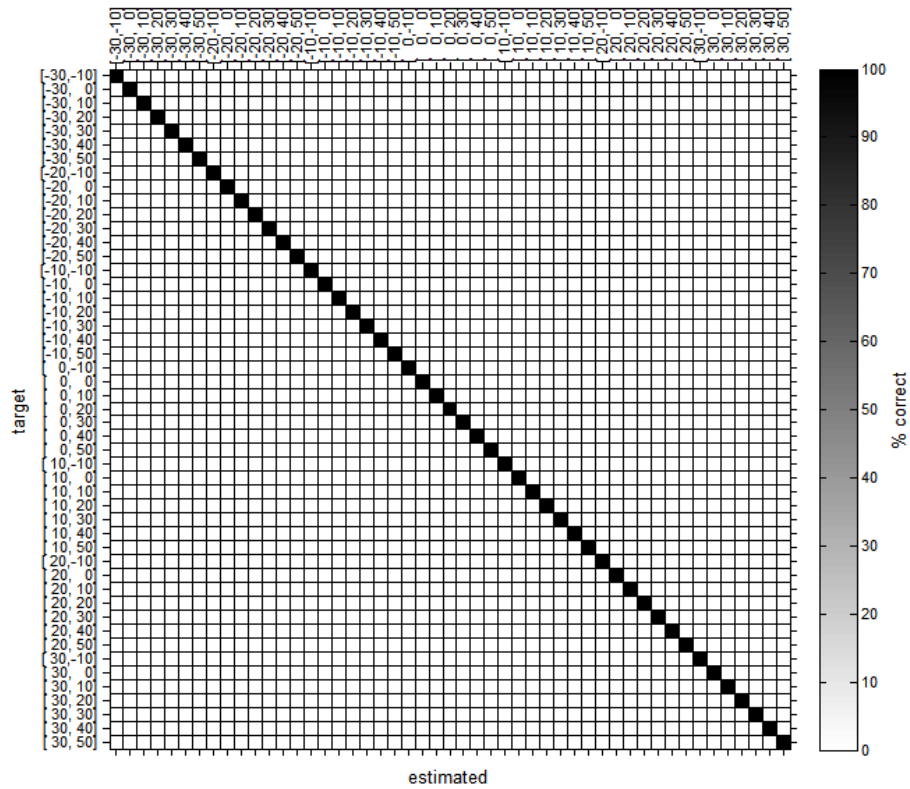


(c) Rauschen; ohne Bottleneck-Vortraining; frontal; sphärisch; ITD



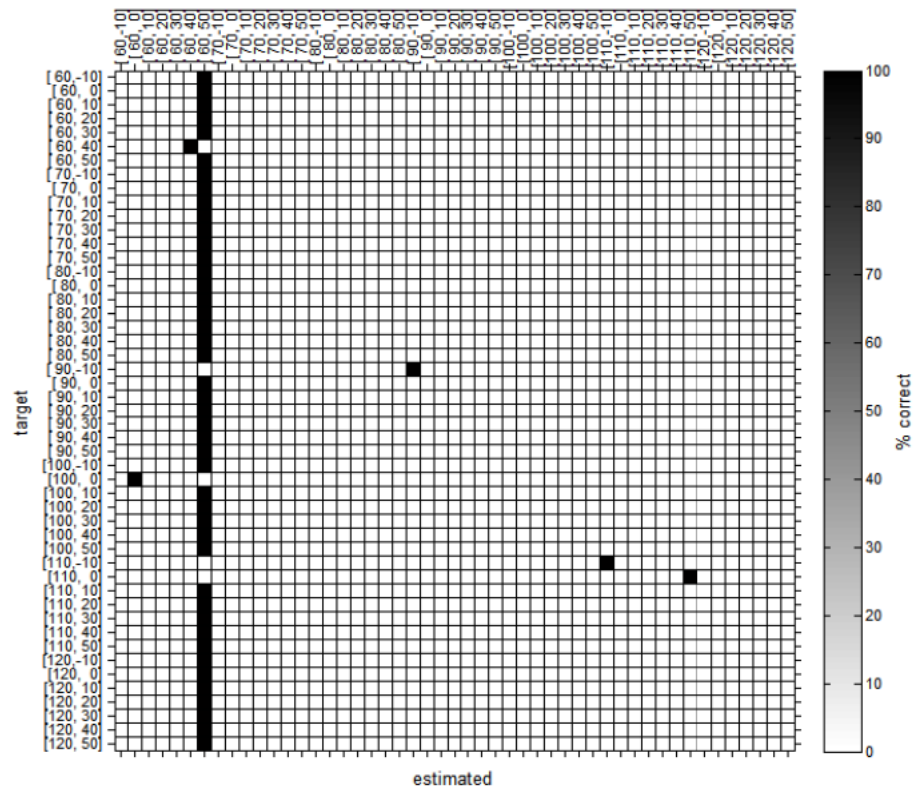
(d) Rauschen; ohne Bottleneck-Vortraining; frontal; sphärisch; Betragsspektrum



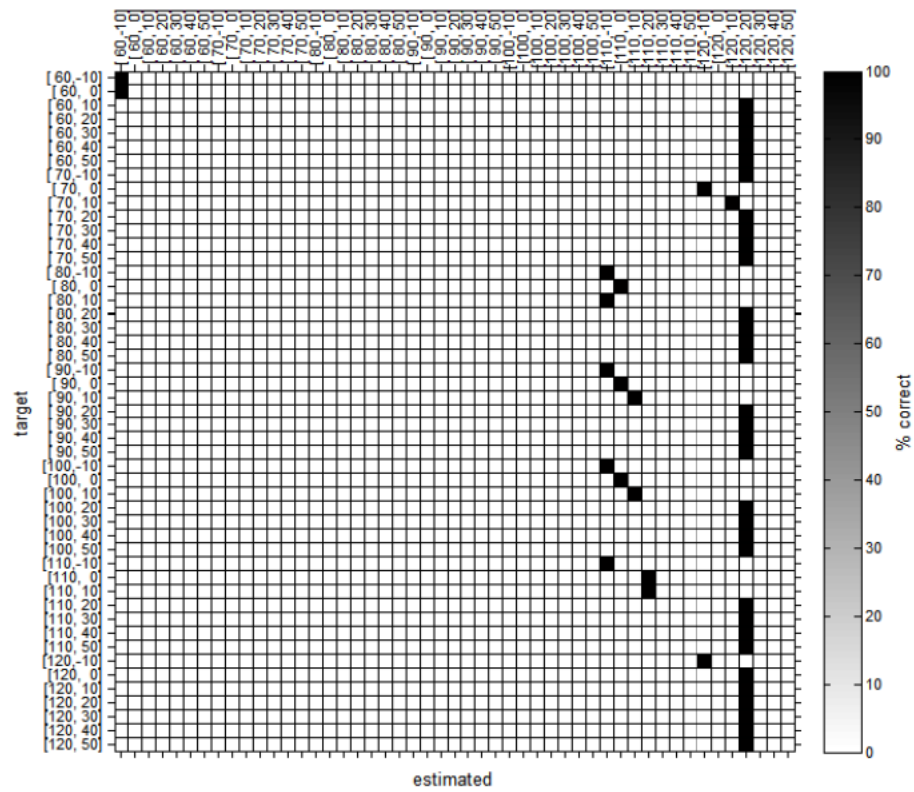


(e) Rauschen; ohne Bottleneck-Vortraining; frontal; sphärisch; **mfcc**

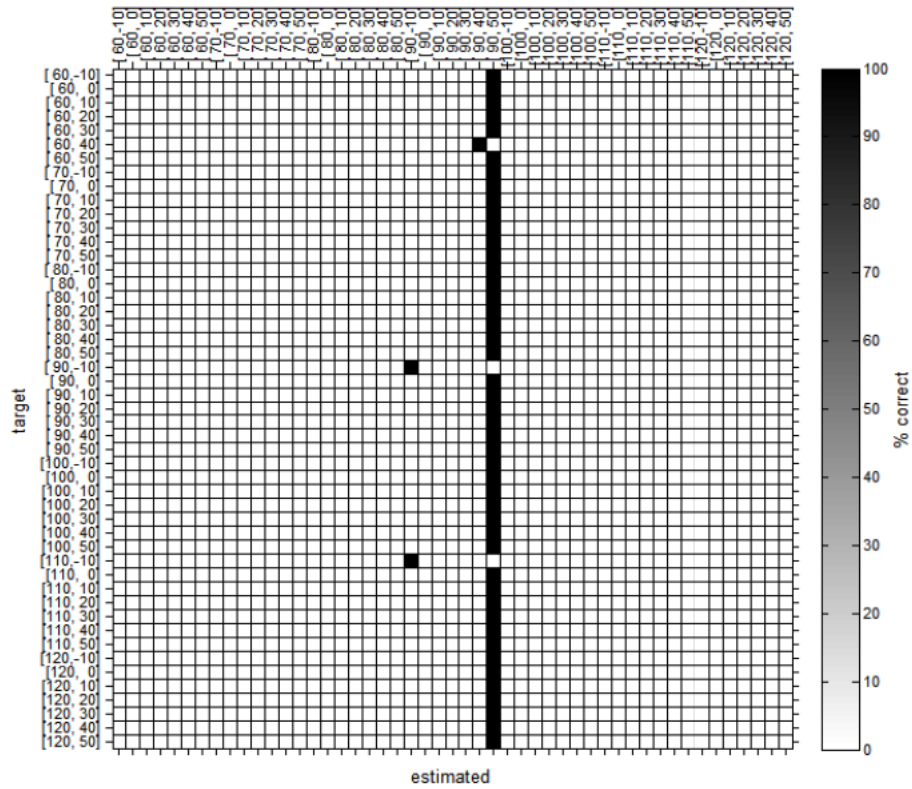
Abbildung A.9: Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; frontal; sphärisch;



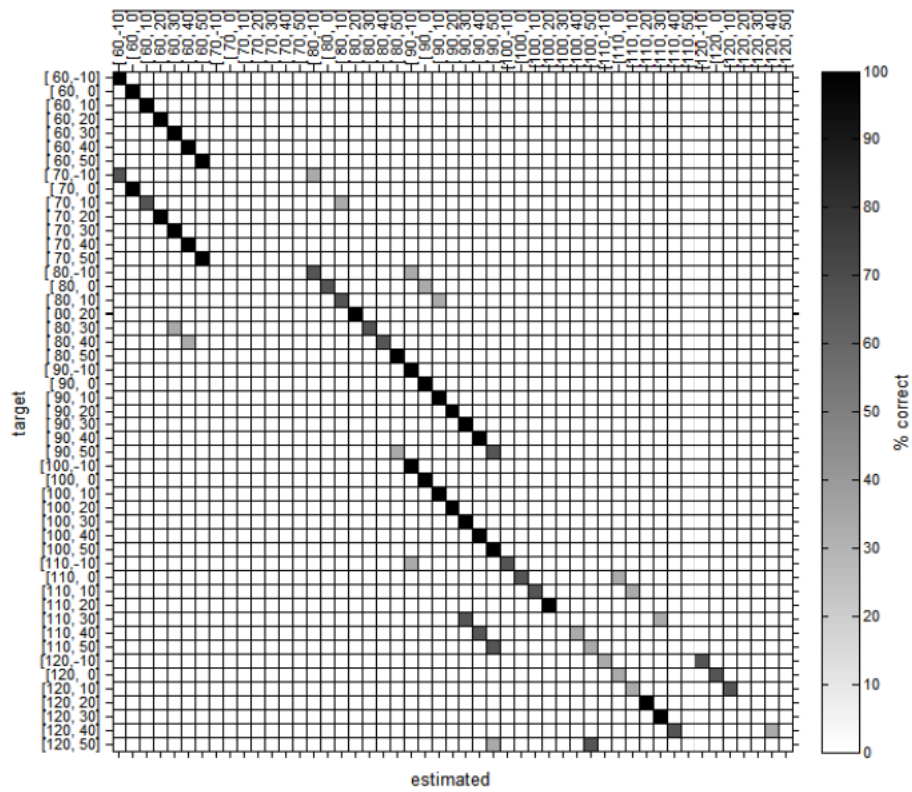
(a) Rauschen; ohne Bottleneck-Vortraining; seitlich; parallel; ILT-ITD



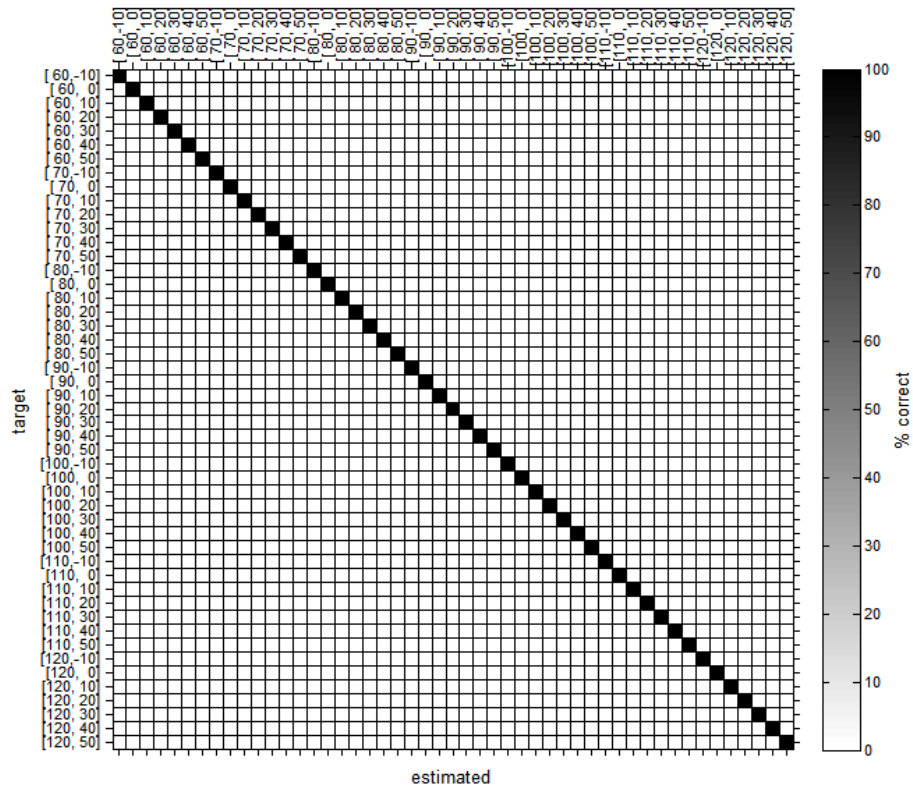
(b) Rauschen; ohne Bottleneck-Vortraining; seitlich; parallel; ILT



(c) Rauschen; ohne Bottleneck-Vortraining; seitlich; parallel; ITD

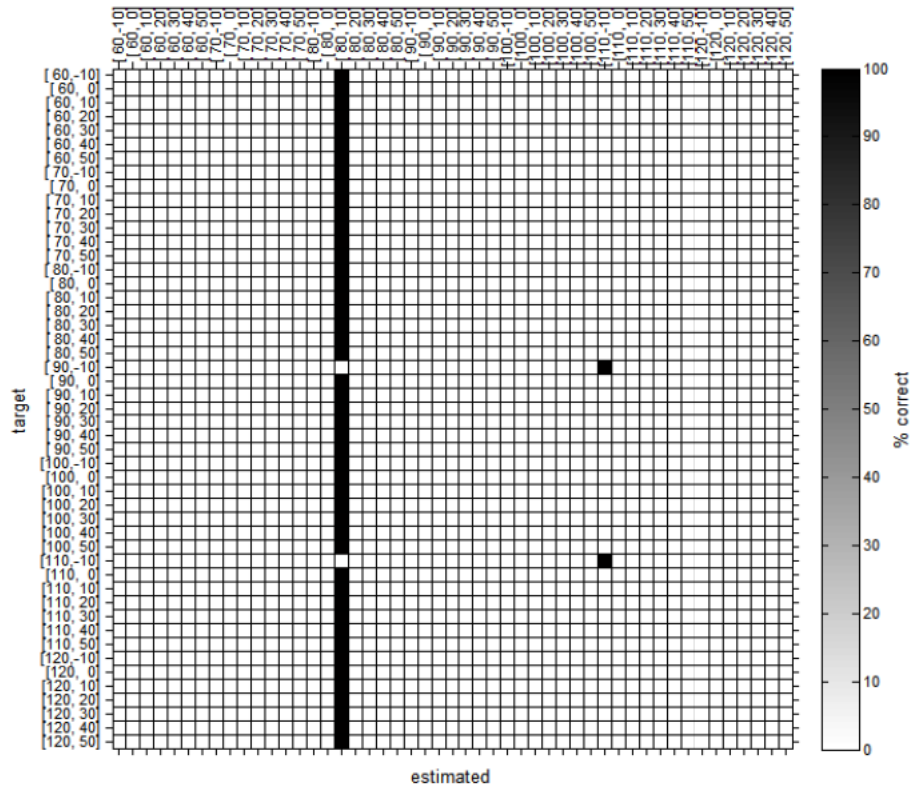


(d) Rauschen; ohne Bottleneck-Vortraining; seitlich; parallel; Betragsspektrum

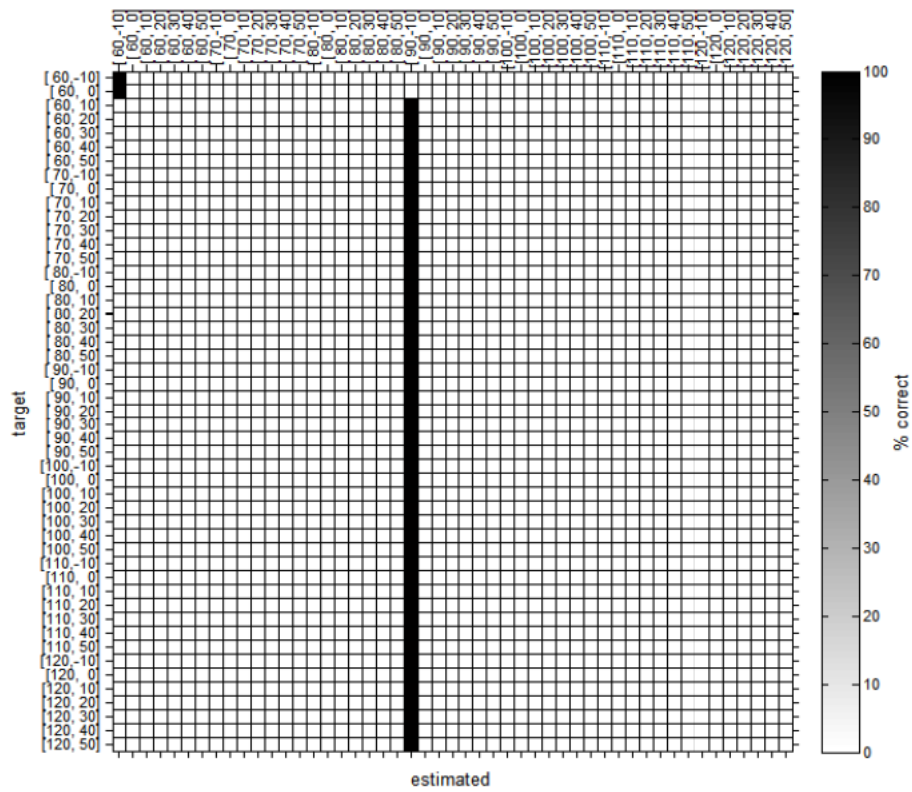


(e) Rauschen; ohne Bottleneck-Vortraining; seitlich; parallel; **mfcc**

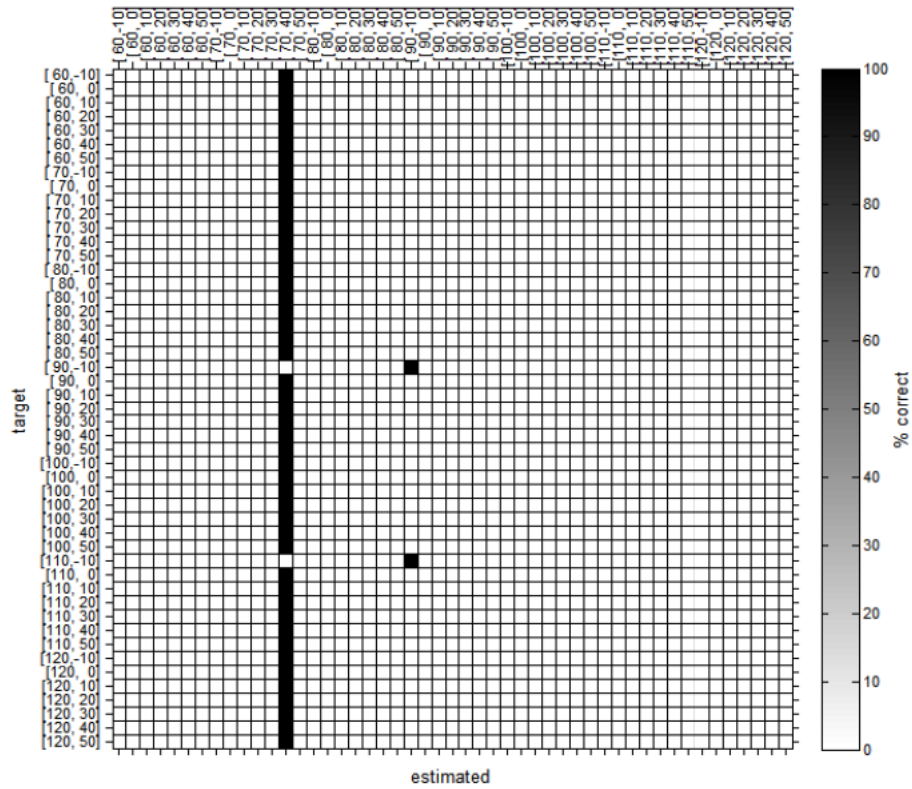
Abbildung A.10: Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; seitlich; parallel;



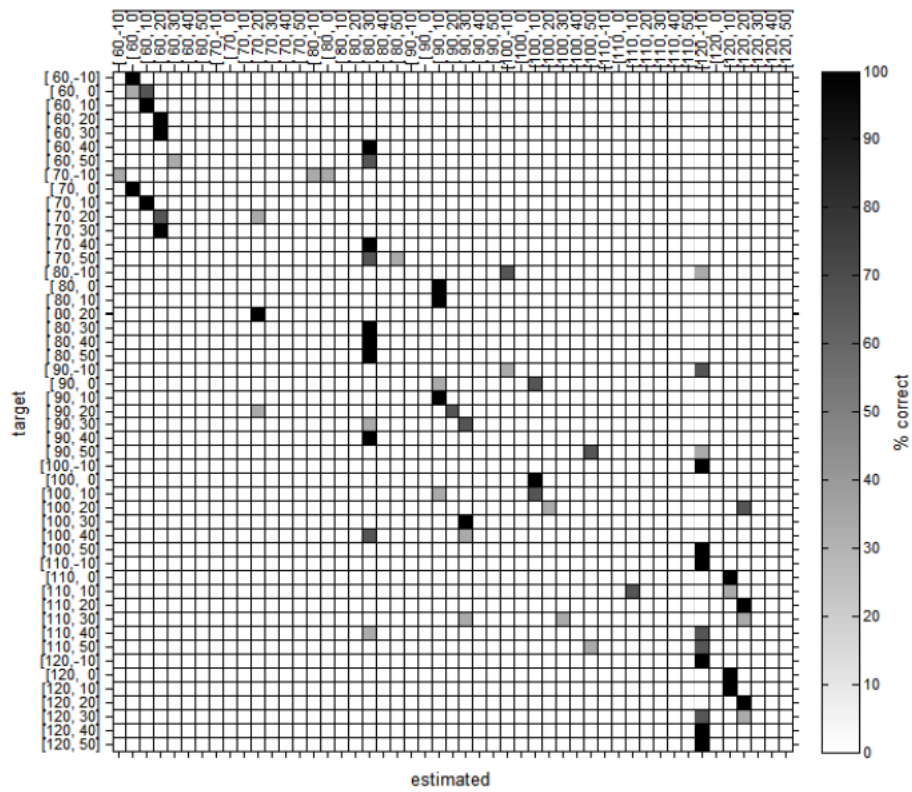
(a) Rauschen; ohne Bottleneck-Vortraining; seitlich; angehängt; ILD-ITD



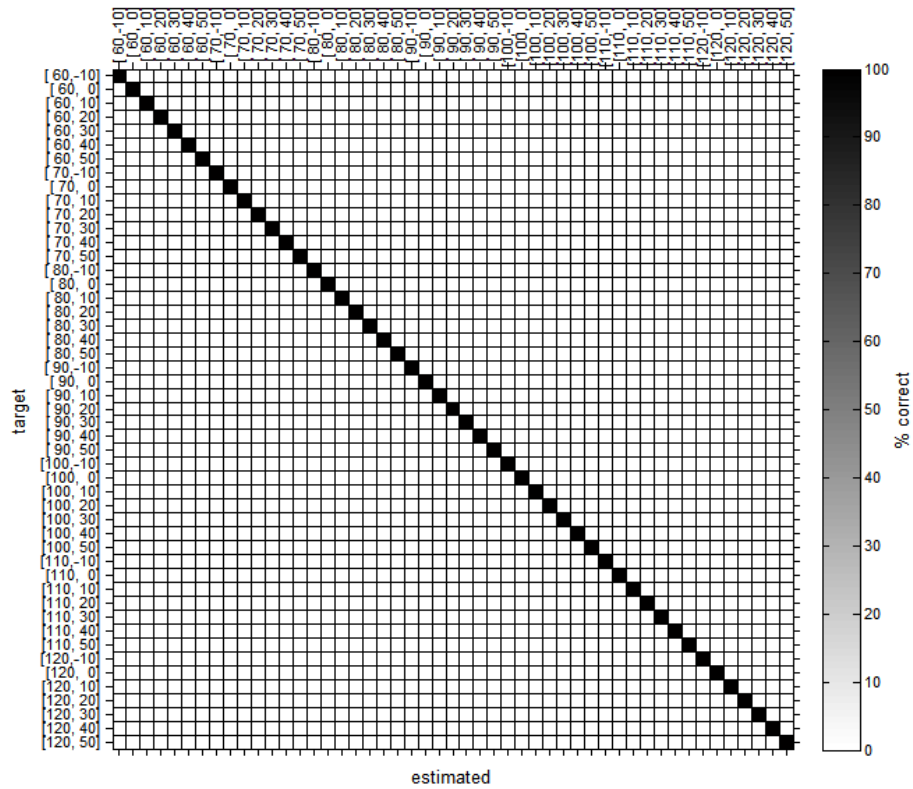
(b) Rauschen; ohne Bottleneck-Vortraining; seitlich; angehängt; ILD



(c) Rauschen; ohne Bottleneck-Vortraining; seitlich; angehängt; ITD



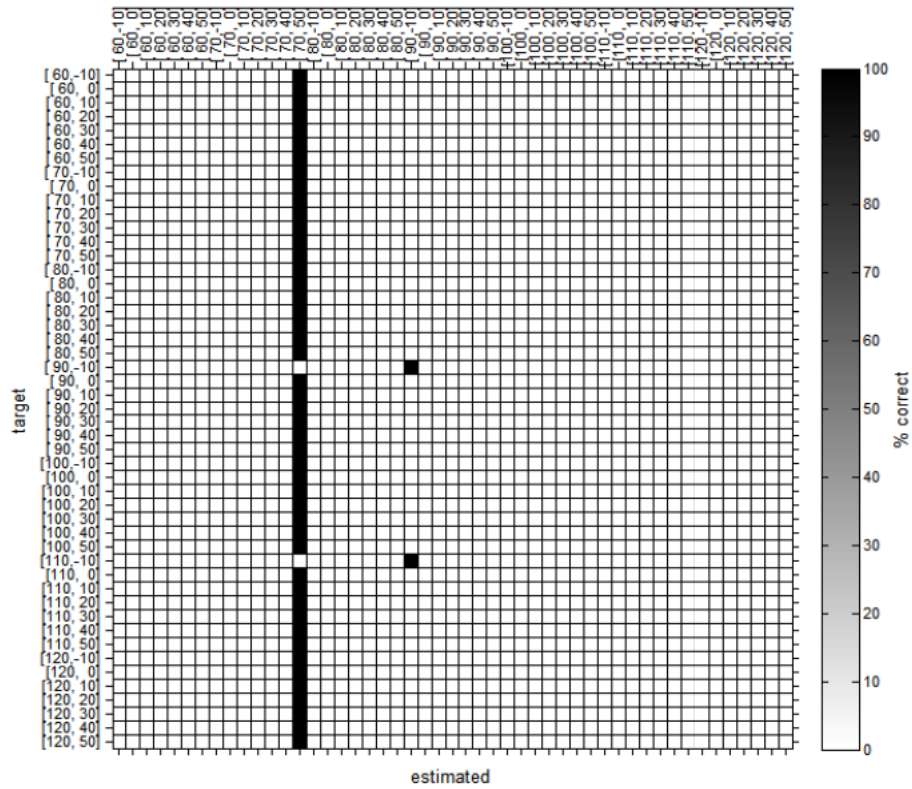
(d) Rauschen; ohne Bottleneck-Vortraining; seitlich; angehängt; Betragsspektrum



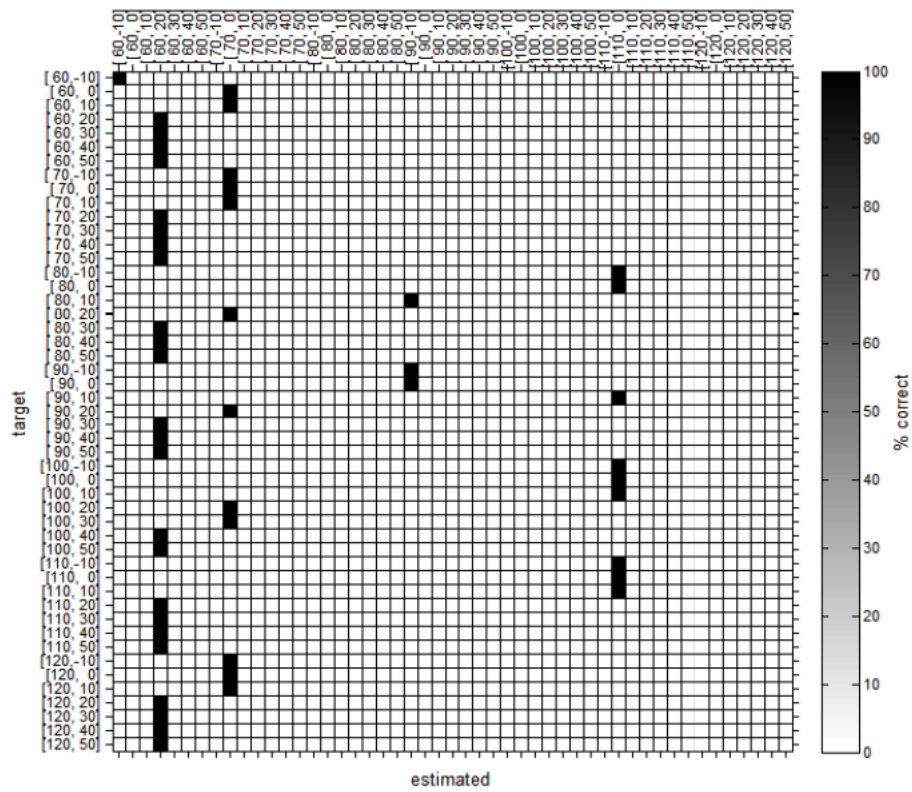
(e) Rauschen; ohne Bottleneck-Vortraining; seitlich; angehängt; **mfcc**

Abbildung A.11: Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; seitlich; angehängt;



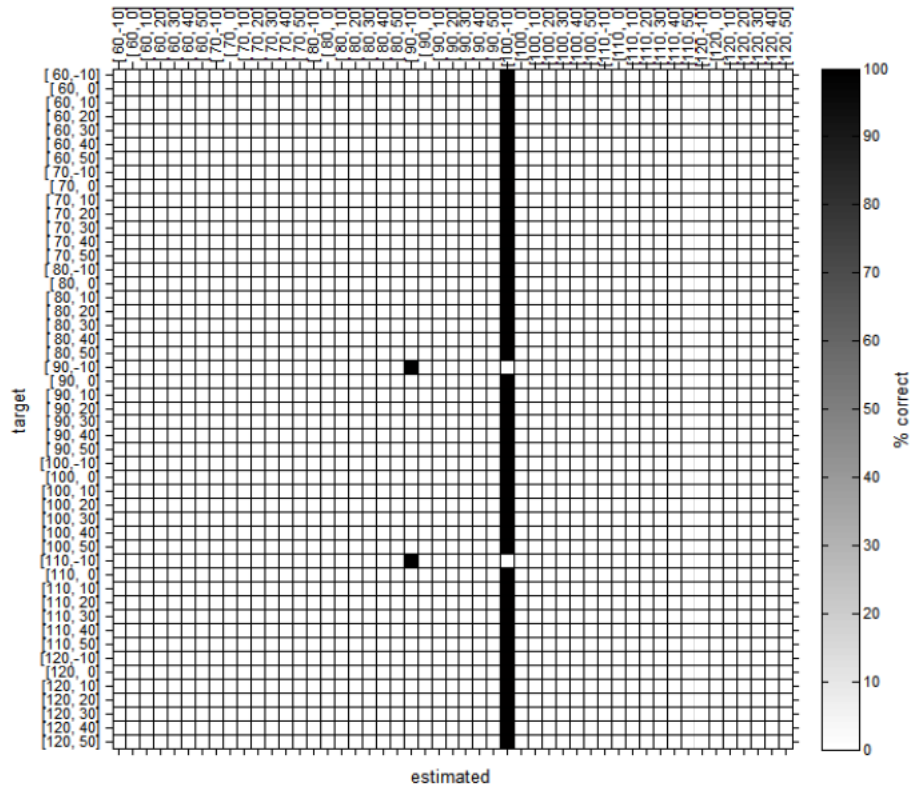


(a) Rauschen; ohne Bottleneck-Vortraining; seitlich; sphärisch; ILD-ITD

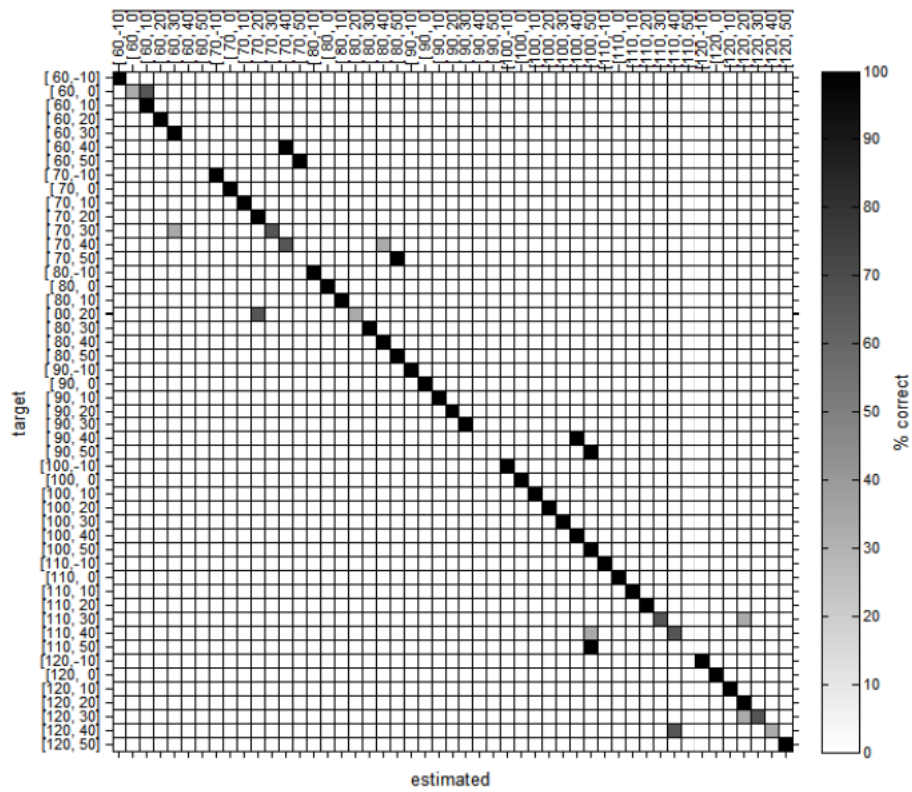


(b) Rauschen; ohne Bottleneck-Vortraining; seitlich; sphärisch; ILD

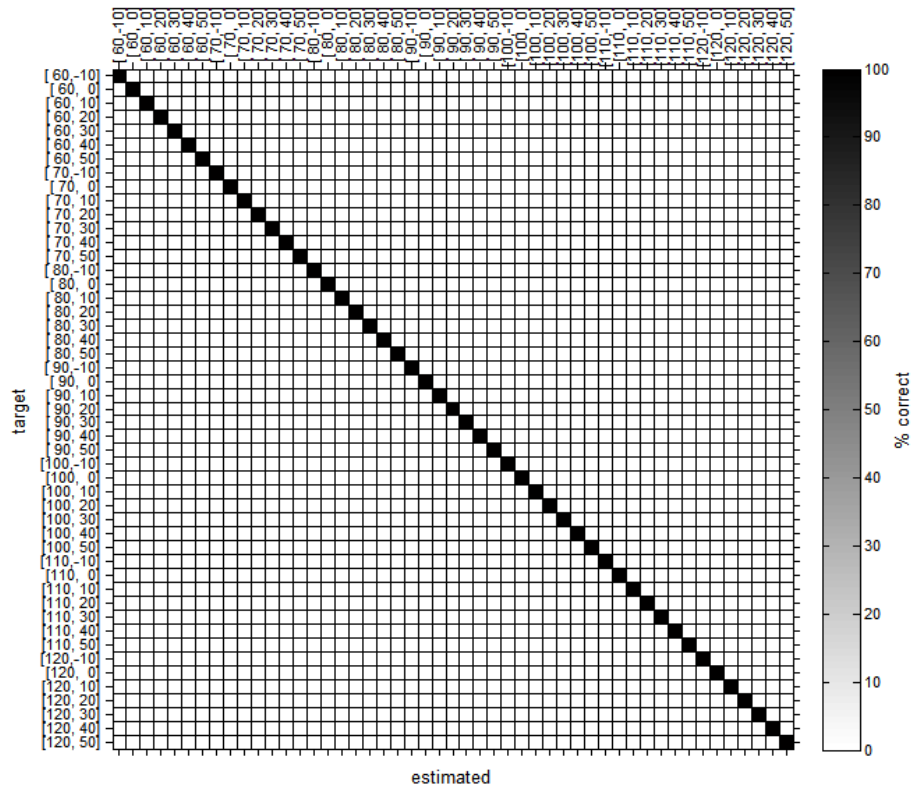




(c) Rauschen; ohne Bottleneck-Vortraining; seitlich; sphärisch; ITD

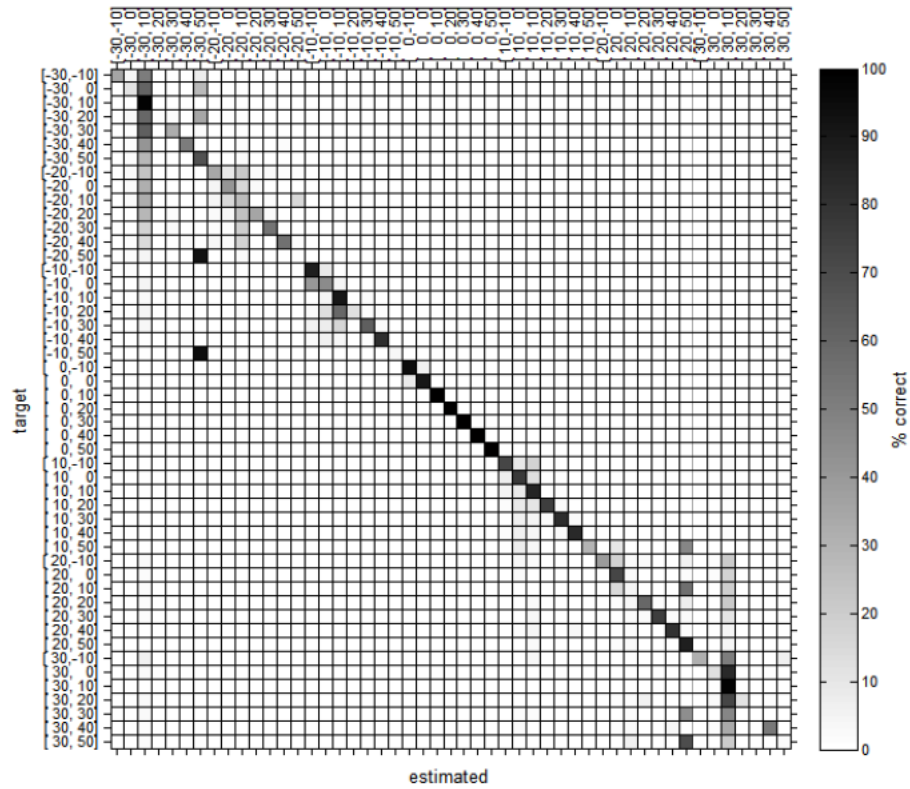


(d) Rauschen; ohne Bottleneck-Vortraining; seitlich; sphärisch; Betragsspektrum

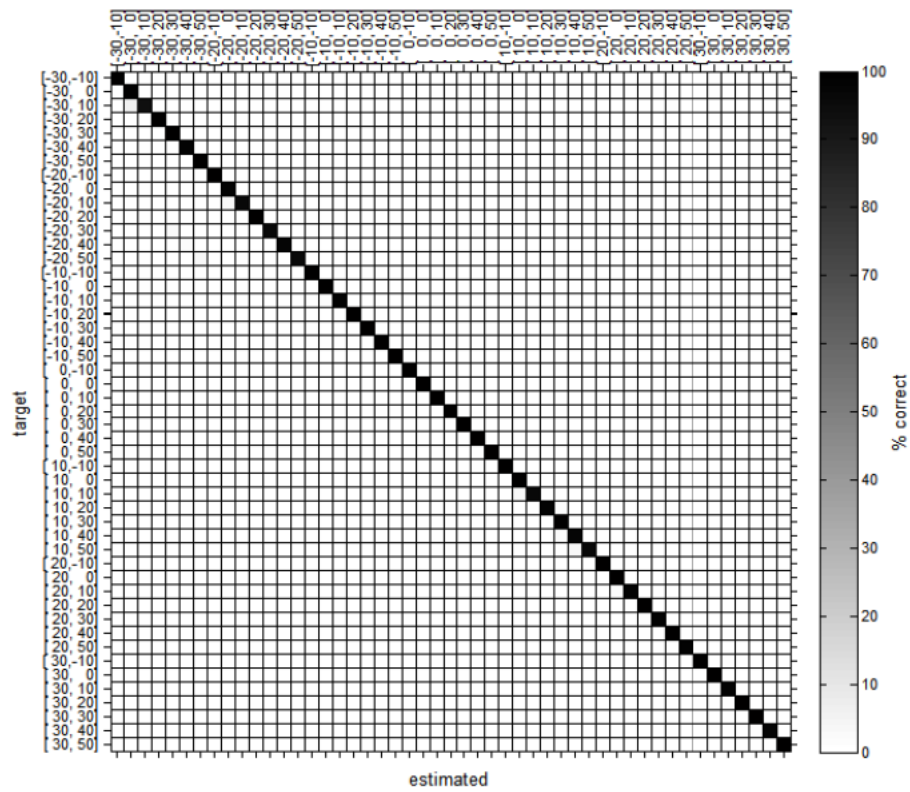


(e) Rauschen; ohne Bottleneck-Vortraining; seitlich; sphärisch; **mfcc**

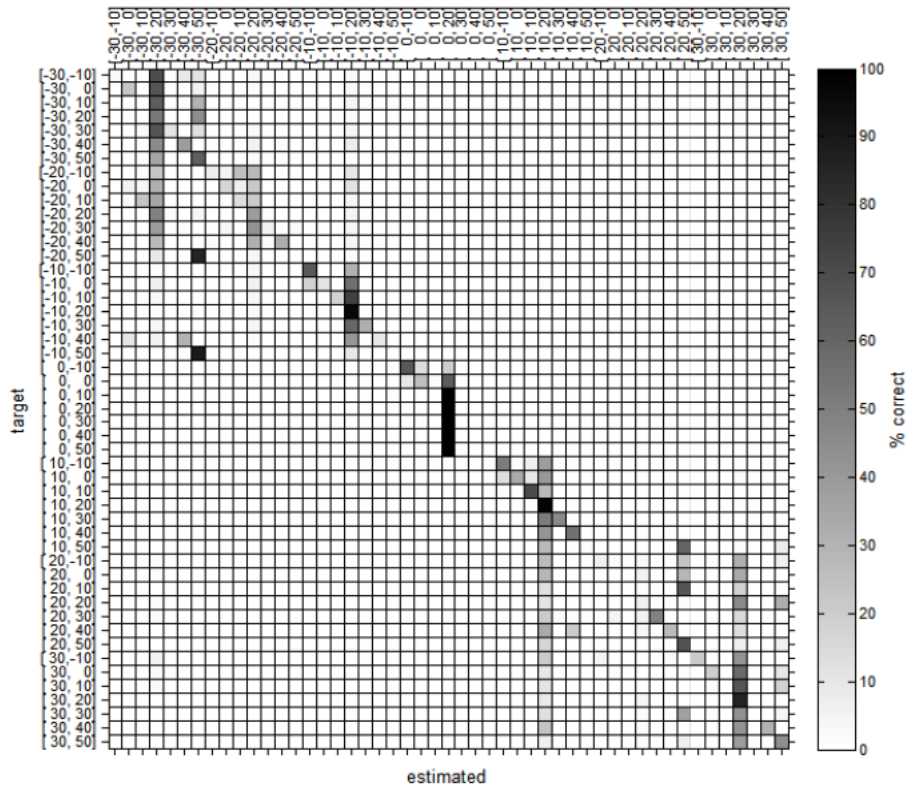
Abbildung A.12: Confusion-Matrizen für die Konditionen: Rauschen; ohne Bottleneck-Vortraining; seitlich; sphärisch;



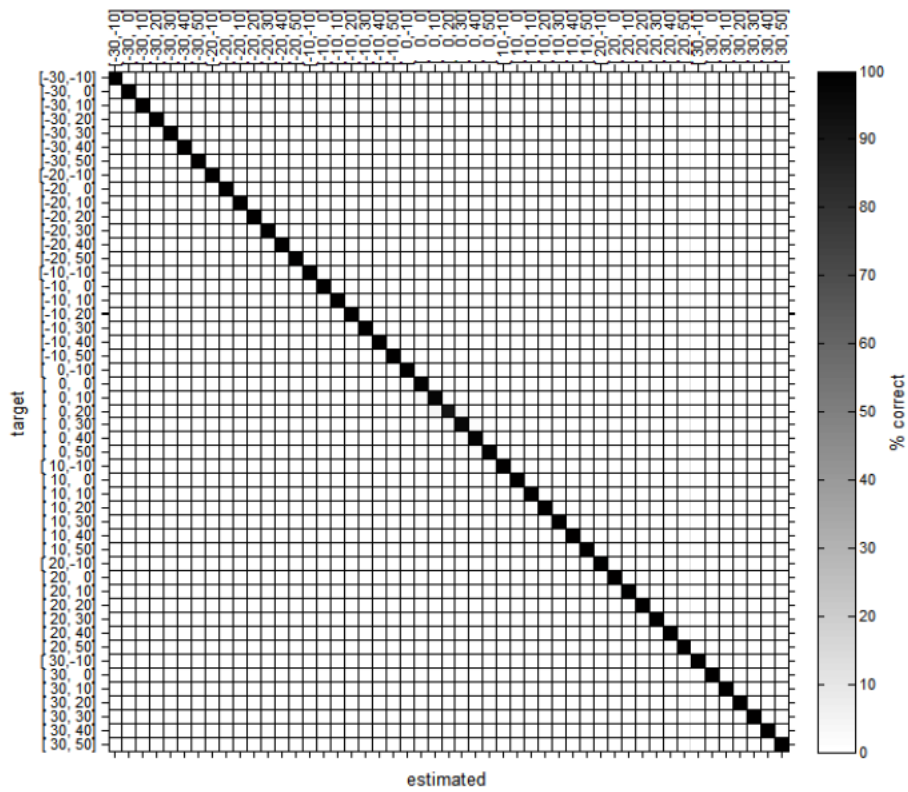
(a) Sprache; Vortraining mit Bottleneck; frontal; parallel; ILD-ITD



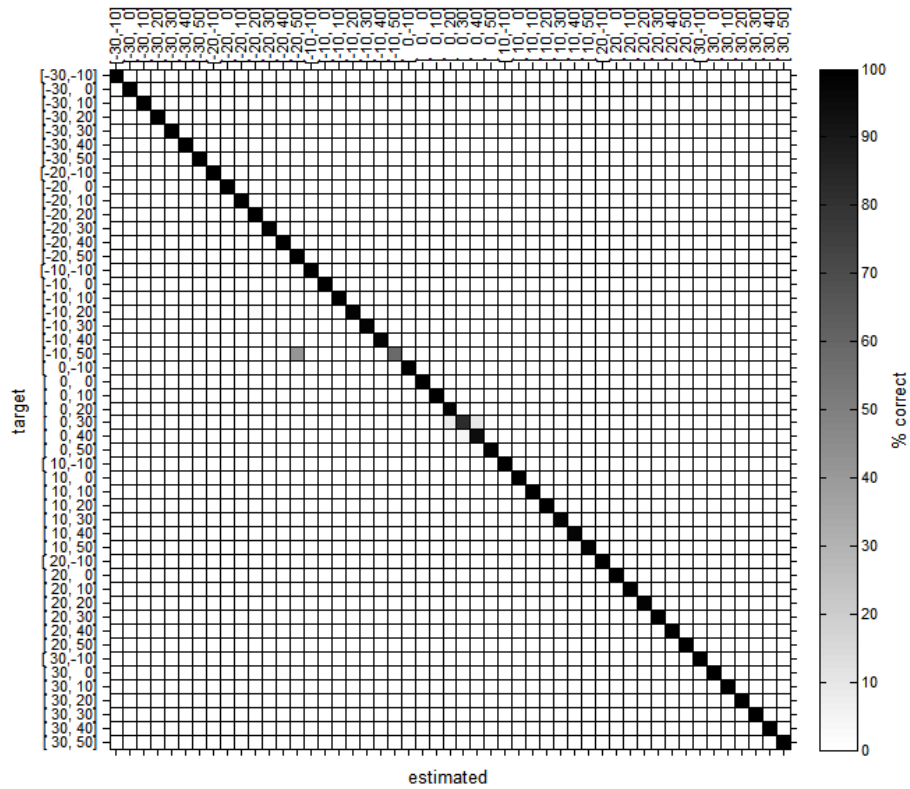
(b) Sprache; Vortraining mit Bottleneck; frontal; parallel; ILD



(c) Sprache; Vortraining mit Bottleneck; frontal; parallel; ITD

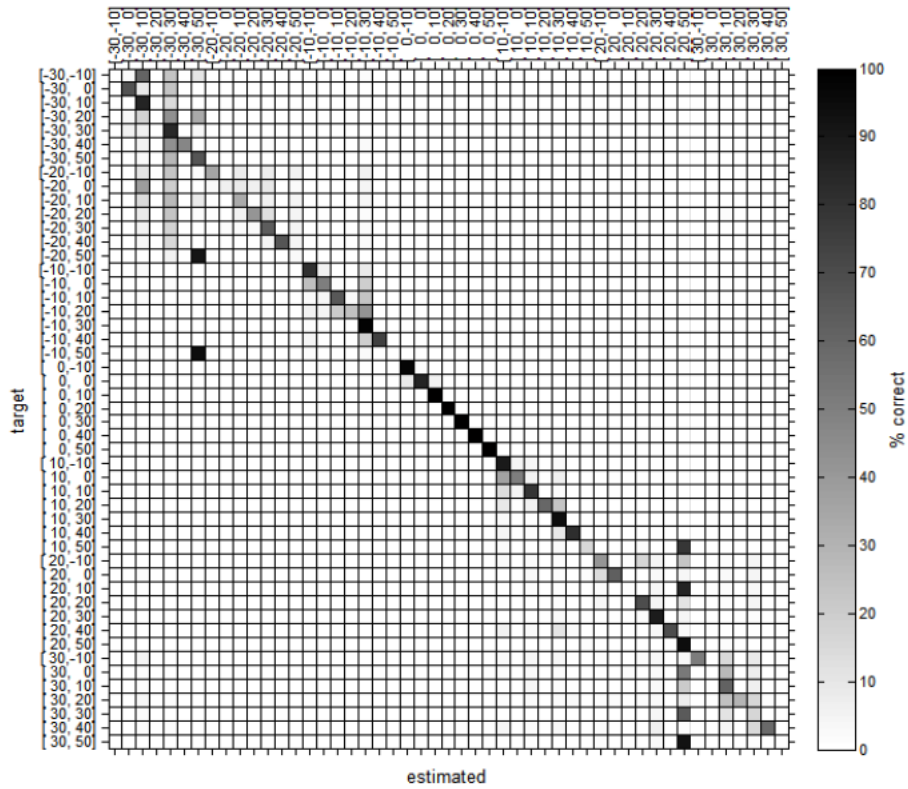


(d) Sprache; Vortraining mit Bottleneck; frontal; parallel; Betragsspektrum

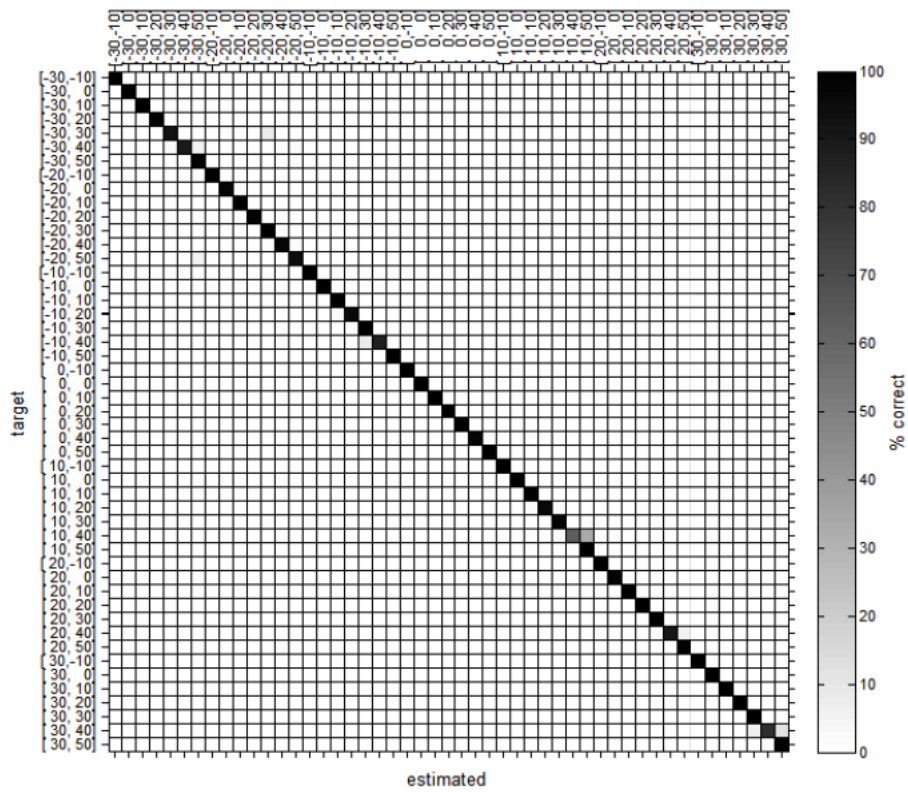


(e) Sprache; Vortraining mit Bottleneck; frontal; parallel; **mfcc**

Abbildung A.13: Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit Bottleneck; frontal; parallel;

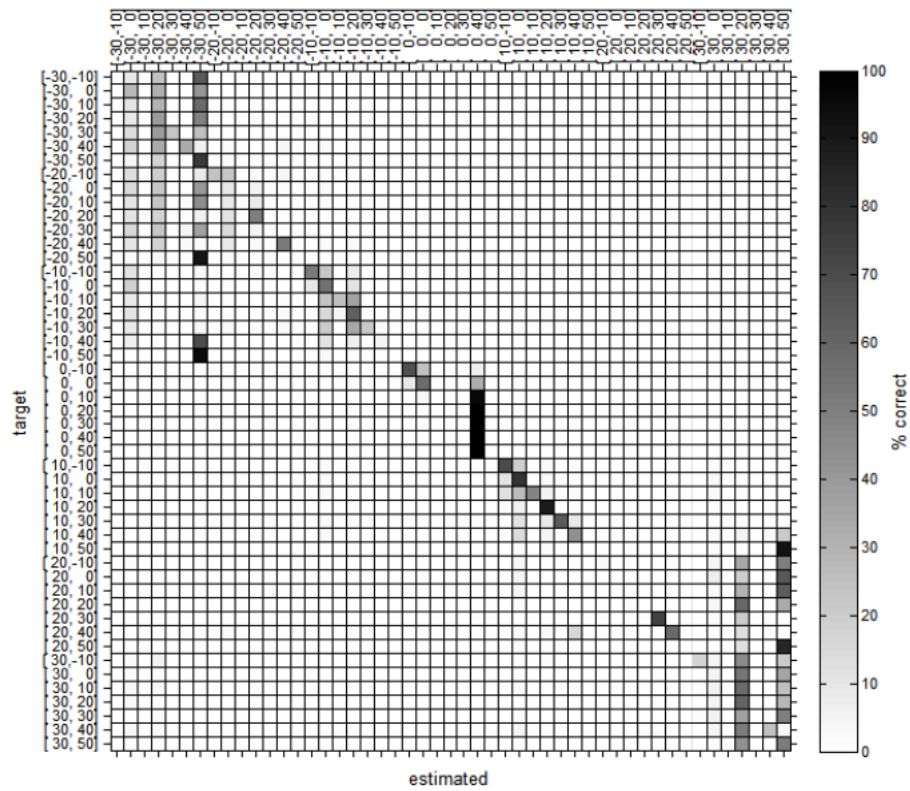


(a) Sprache; Vortraining mit Bottleneck; frontal; angehängt; ILD-ITD

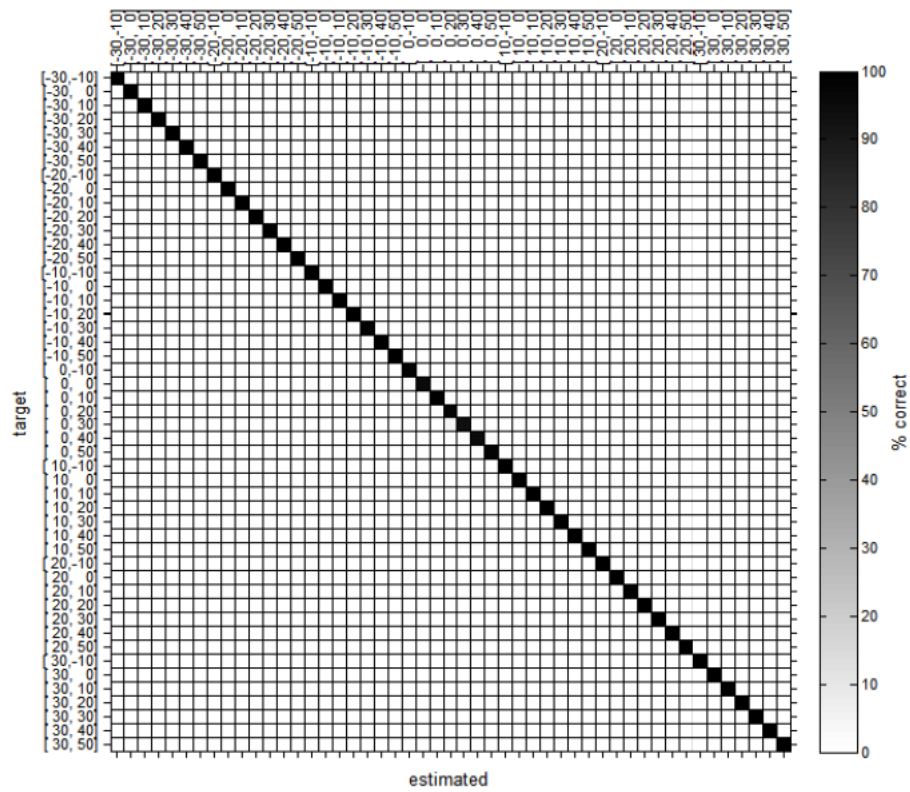


(b) Sprache; Vortraining mit Bottleneck; frontal; angehängt; ILD

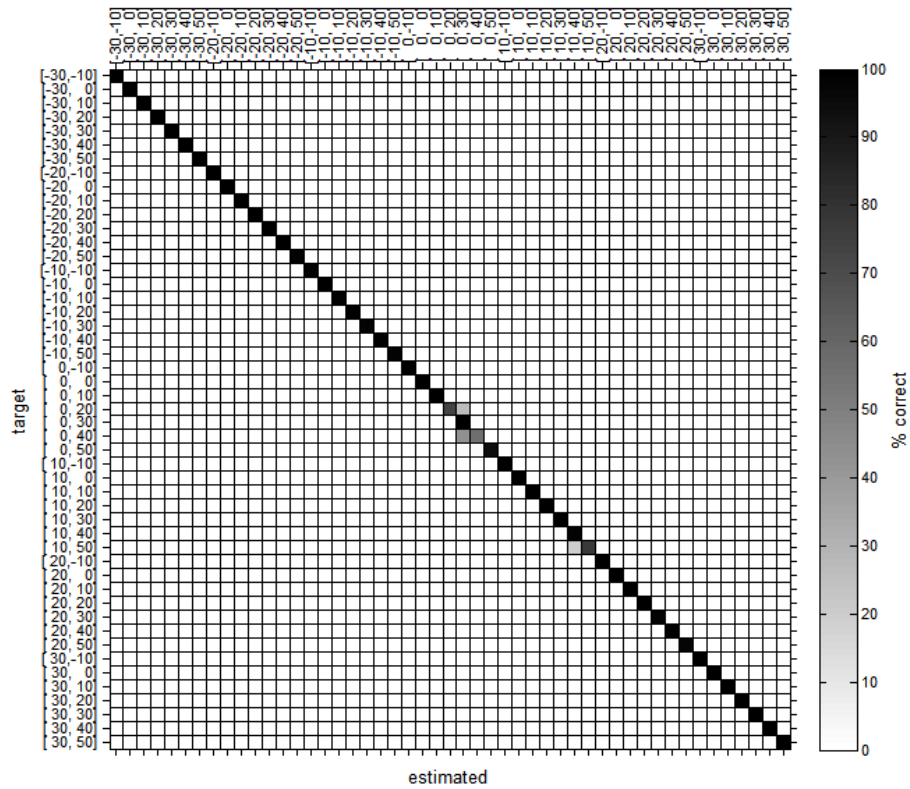




(c) az-el-attached ITD-narrow



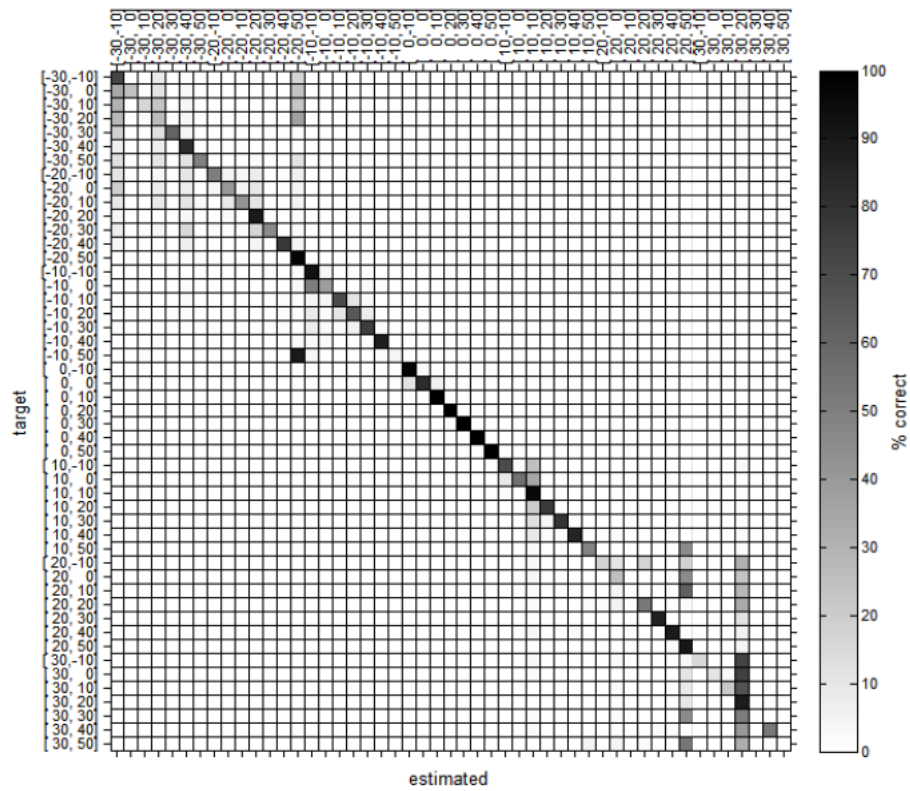
(d) Sprache; Vortraining mit Bottleneck; frontal; angehängt; Betragsspektrum



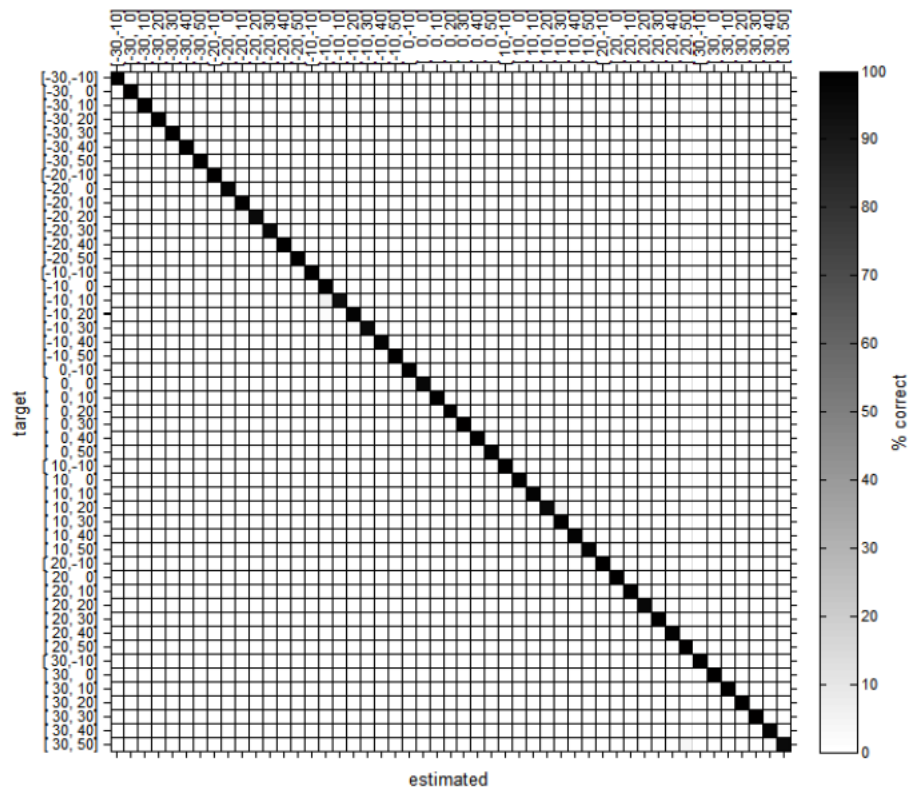
(e) Sprache; Vortraining mit Bottleneck; frontal; angehängt; **mfcc**

Abbildung A.14: Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit Bottleneck; frontal; angehängt;

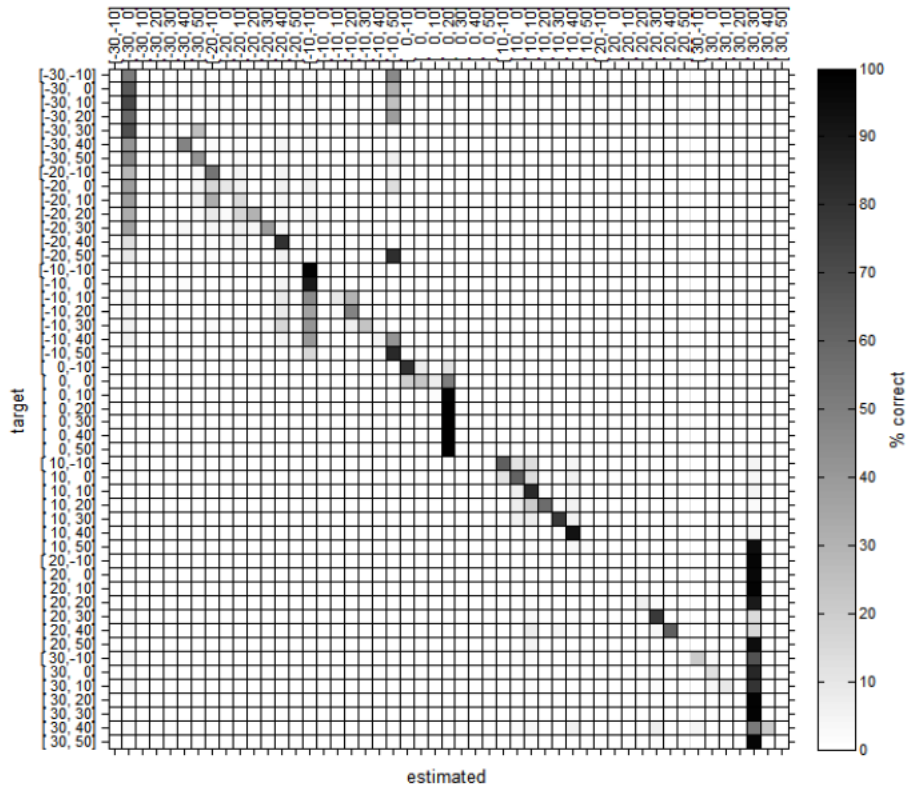




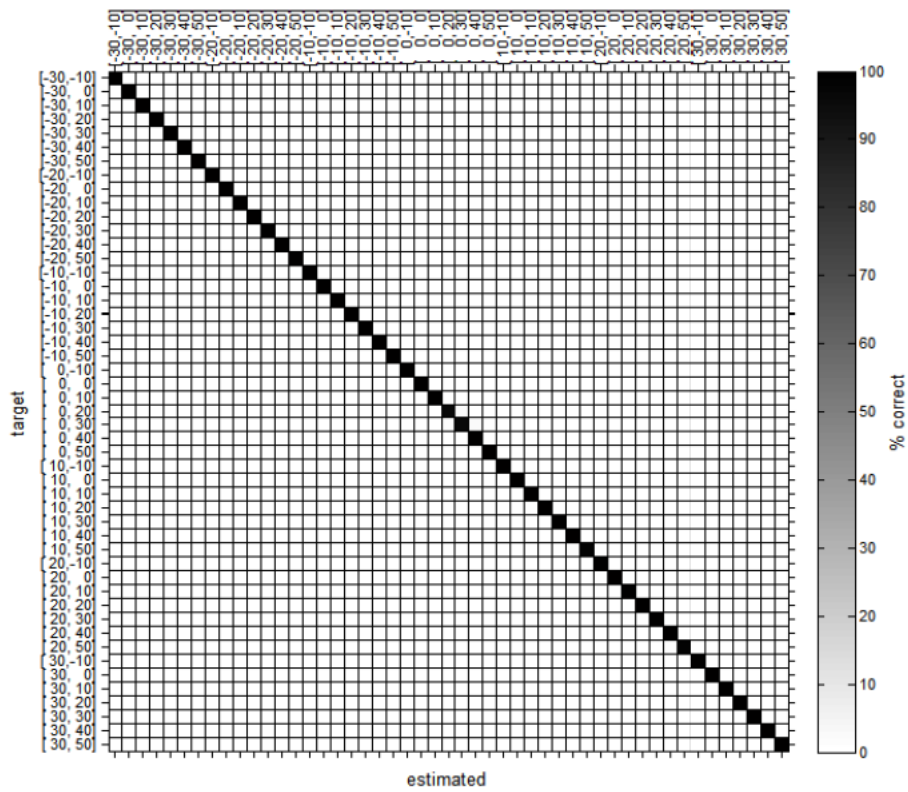
(a) Sprache; Vortraining mit Bottleneck; frontal; sphärisch; ILD-ITD



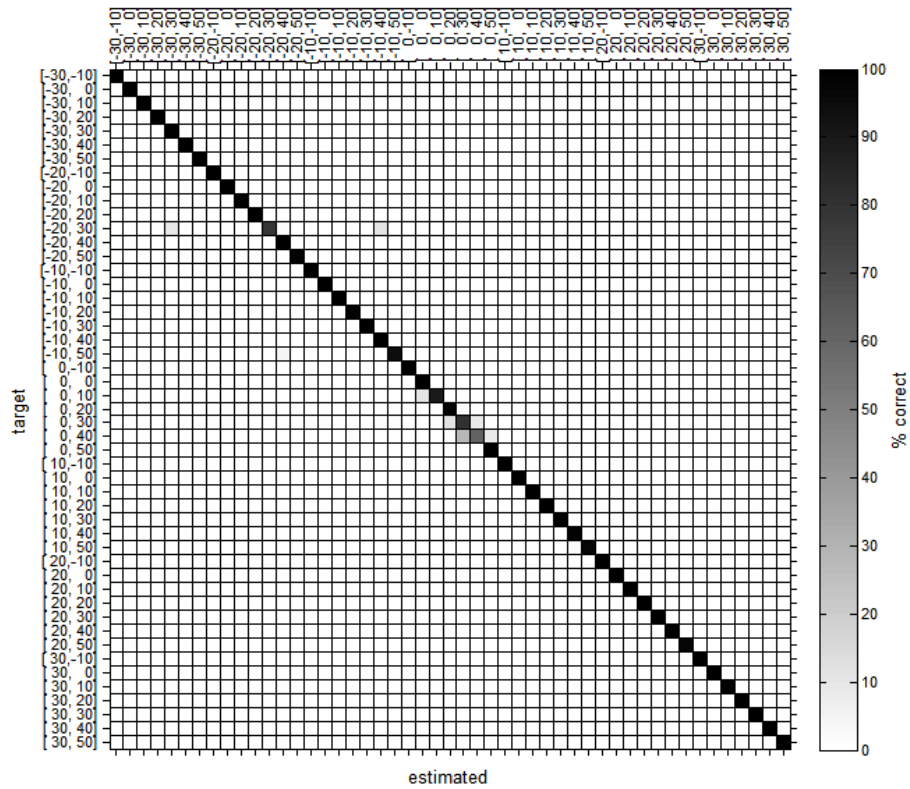
(b) Sprache; Vortraining mit Bottleneck; frontal; sphärisch; ILD



(c) Sprache; Vortraining mit Bottleneck; frontal; sphärisch; ITD

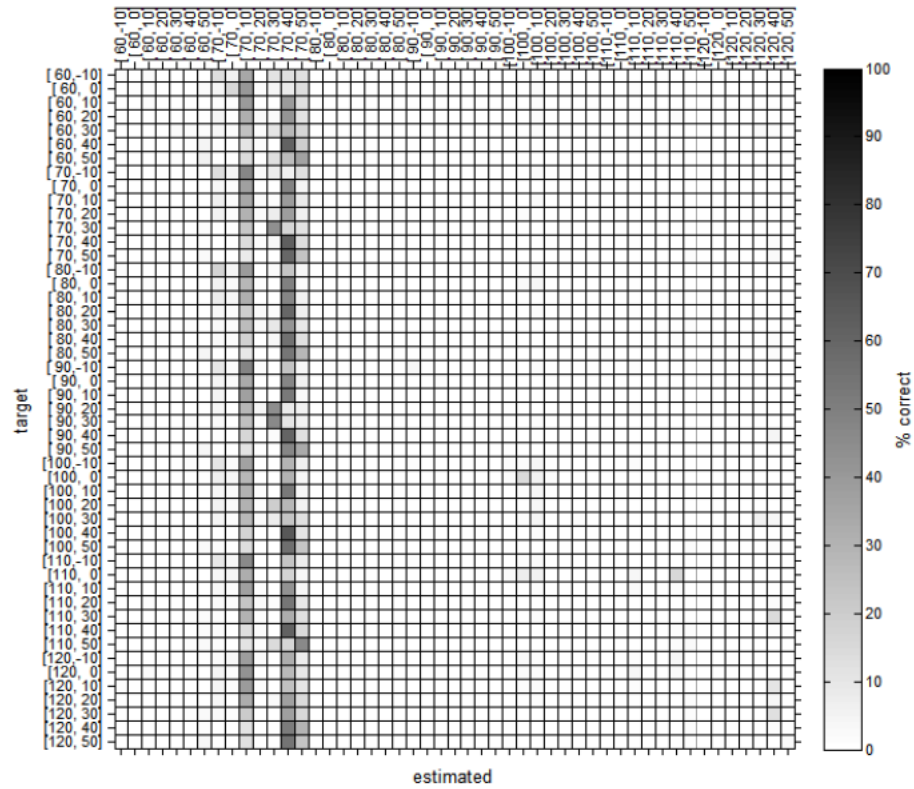


(d) Sprache; Vortraining mit Bottleneck; frontal; sphärisch; Betragsspektrum

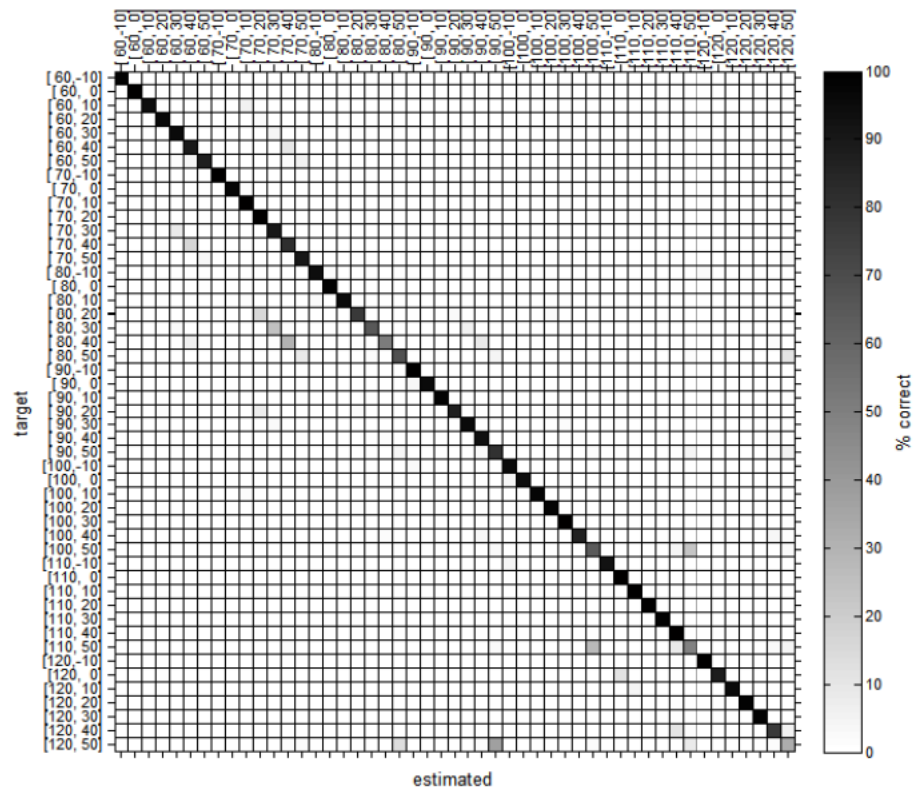


(e) Sprache; Vortraining mit Bottleneck; frontal; sphärisch; **mfcc**

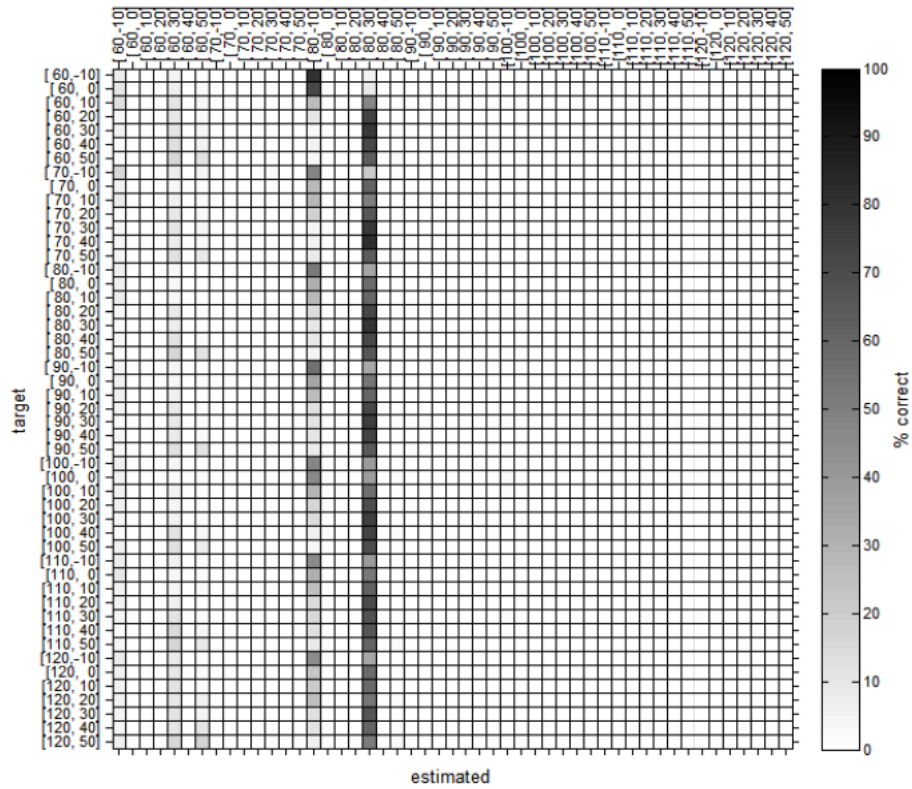
Abbildung A.15: Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit Bottleneck; frontal; sphärisch;



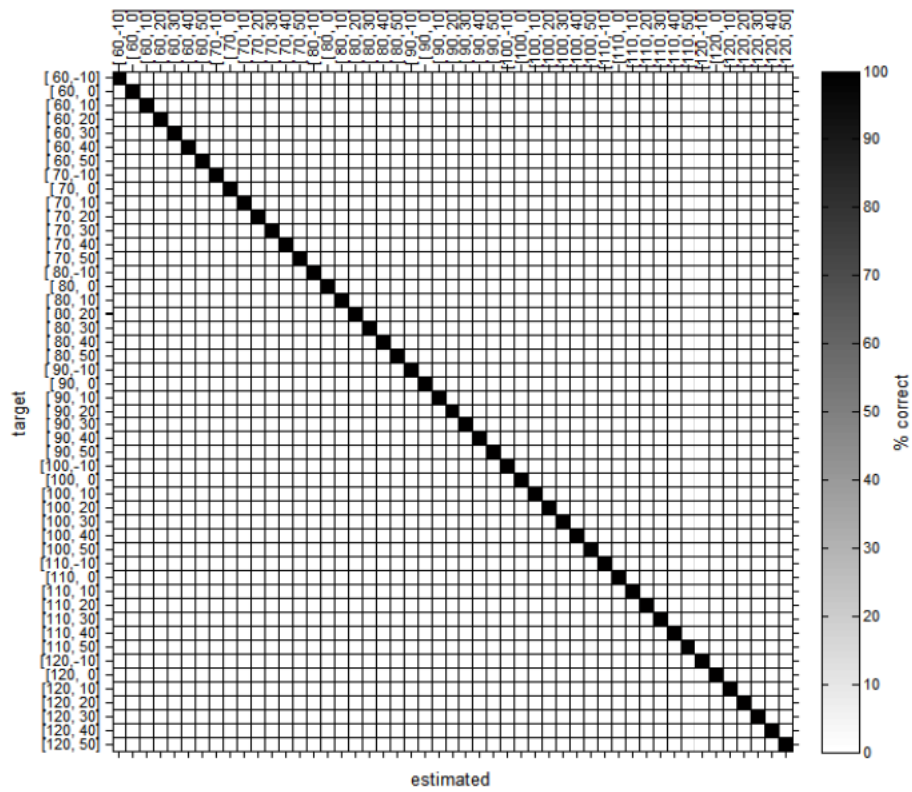
(a) Sprache; Vortraining mit Bottleneck; seitlich; parallel; ILD-ITD



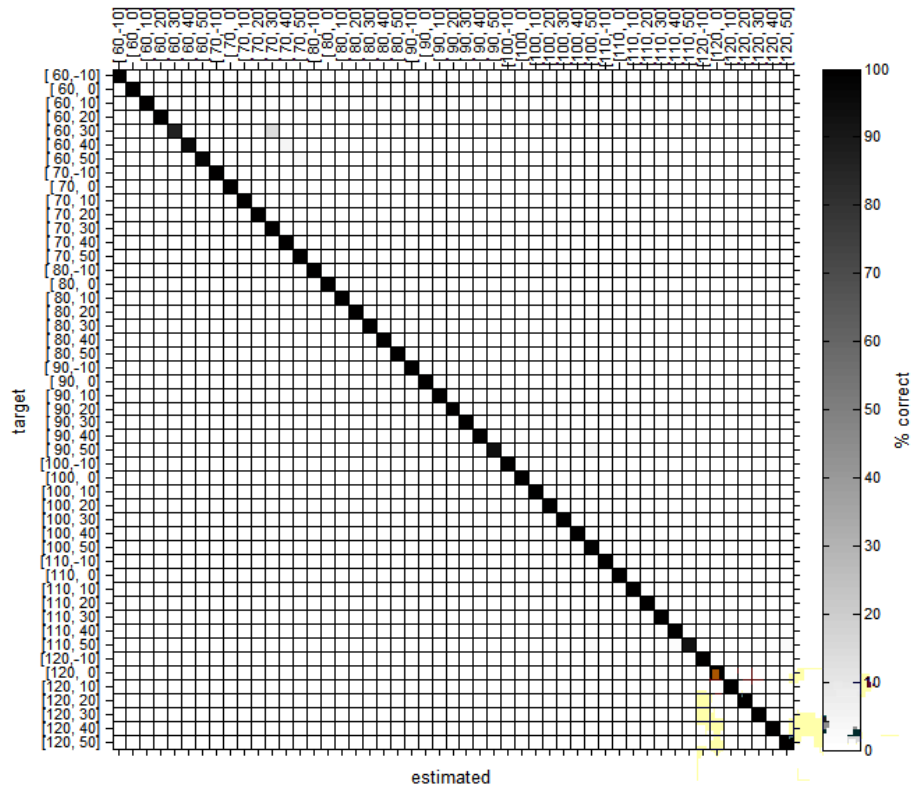
(b) Sprache; Vortraining mit Bottleneck; seitlich; parallel; ILD



(c) Sprache; Vortraining mit Bottleneck; seitlich; parallel; ITD



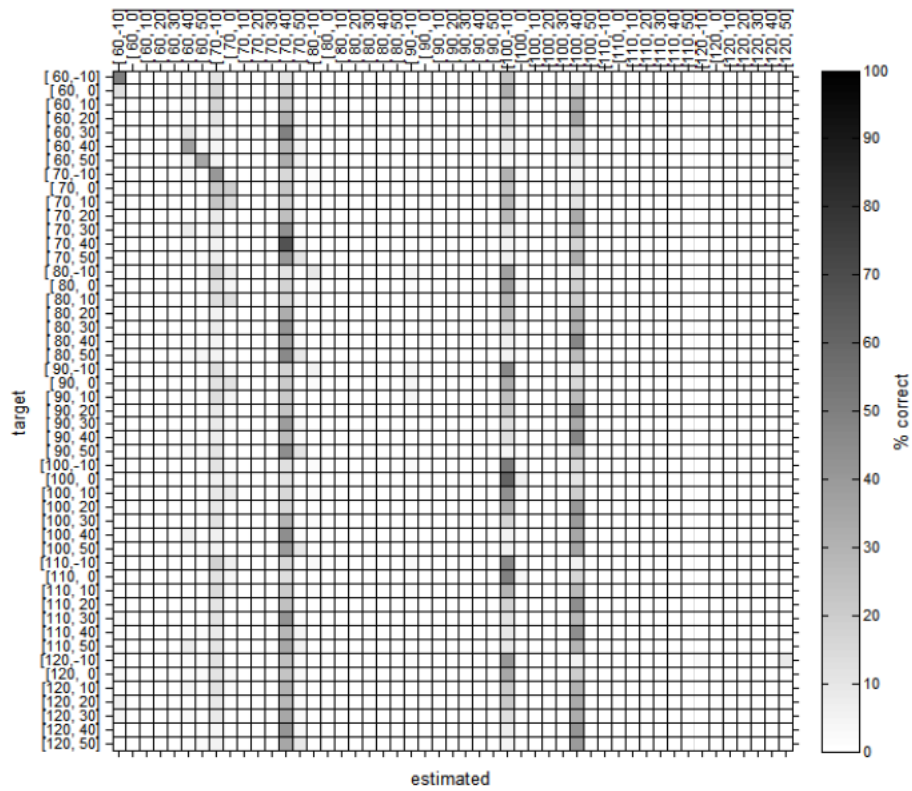
(d) Sprache; Vortraining mit Bottleneck; seitlich; parallel; Betragsspektrum



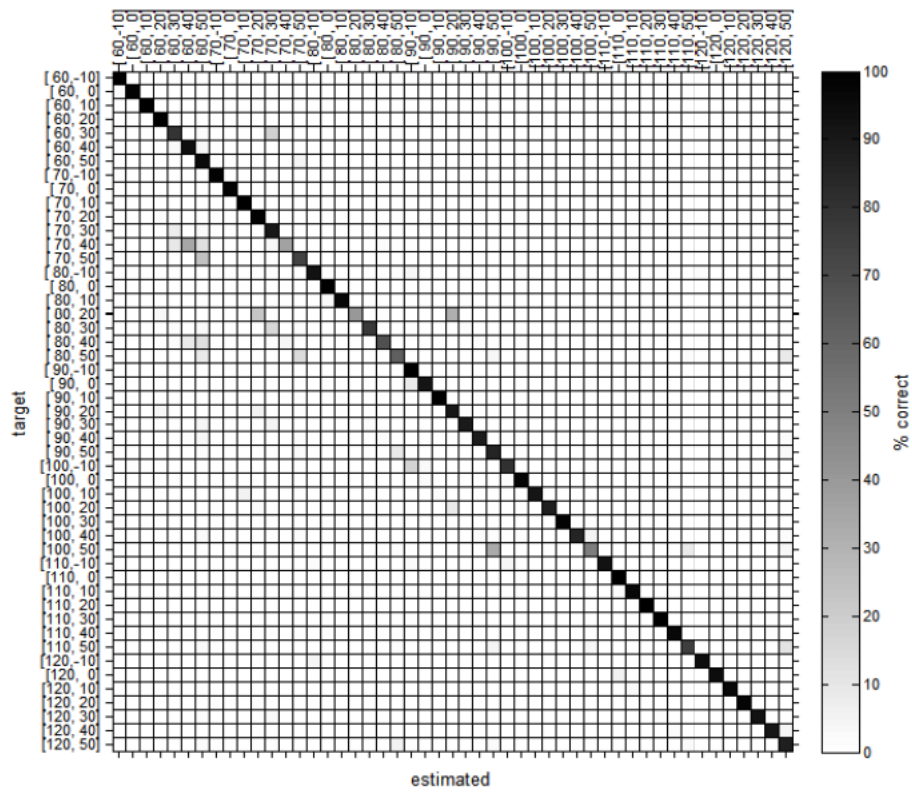
(e) Sprache; Vortraining mit Bottleneck; seitlich; parallel; **mfcc**

Abbildung A.16: Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit Bottleneck; seitlich; parallel;

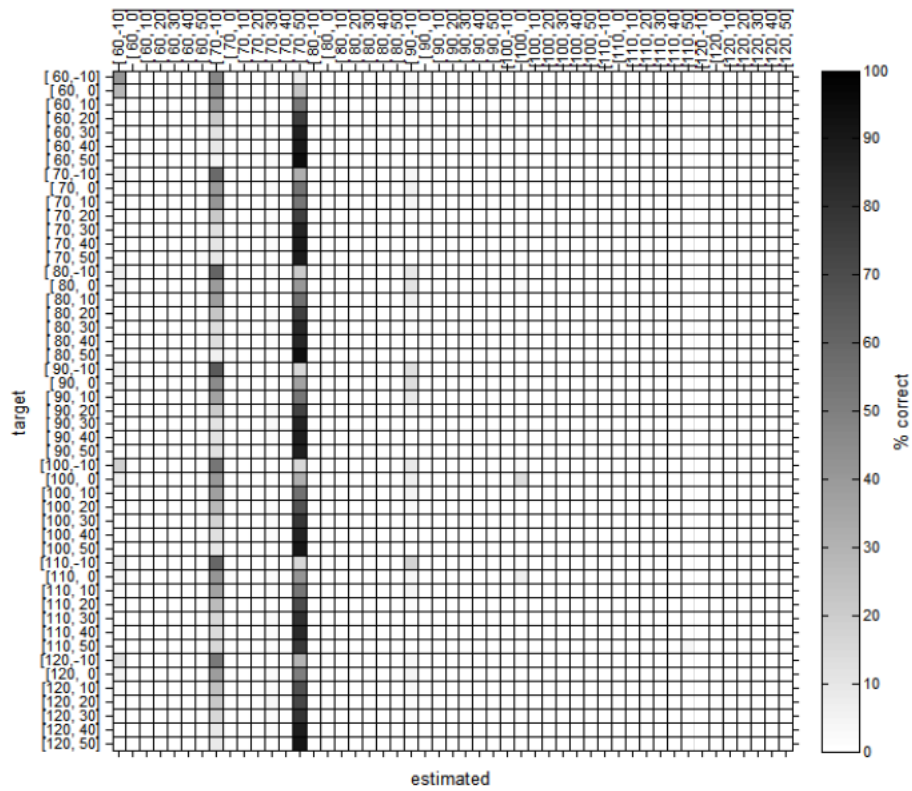




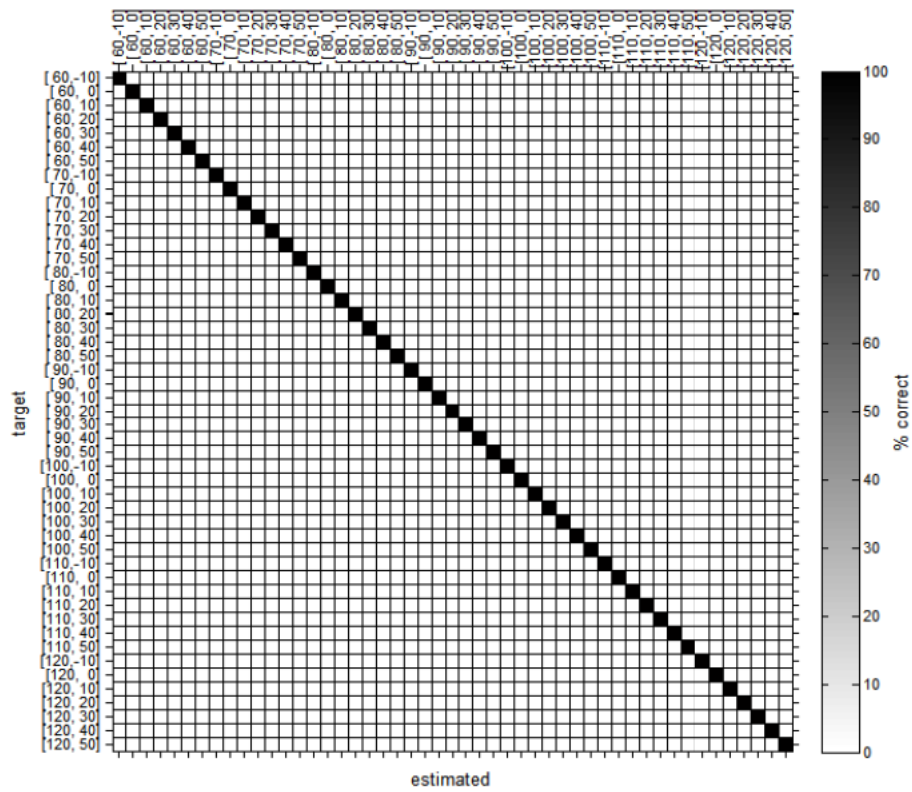
(a) Sprache; Vortraining mit Bottleneck; seitlich; angehängt; ILD-ITD



(b) Sprache; Vortraining mit Bottleneck; seitlich; angehängt; ILD

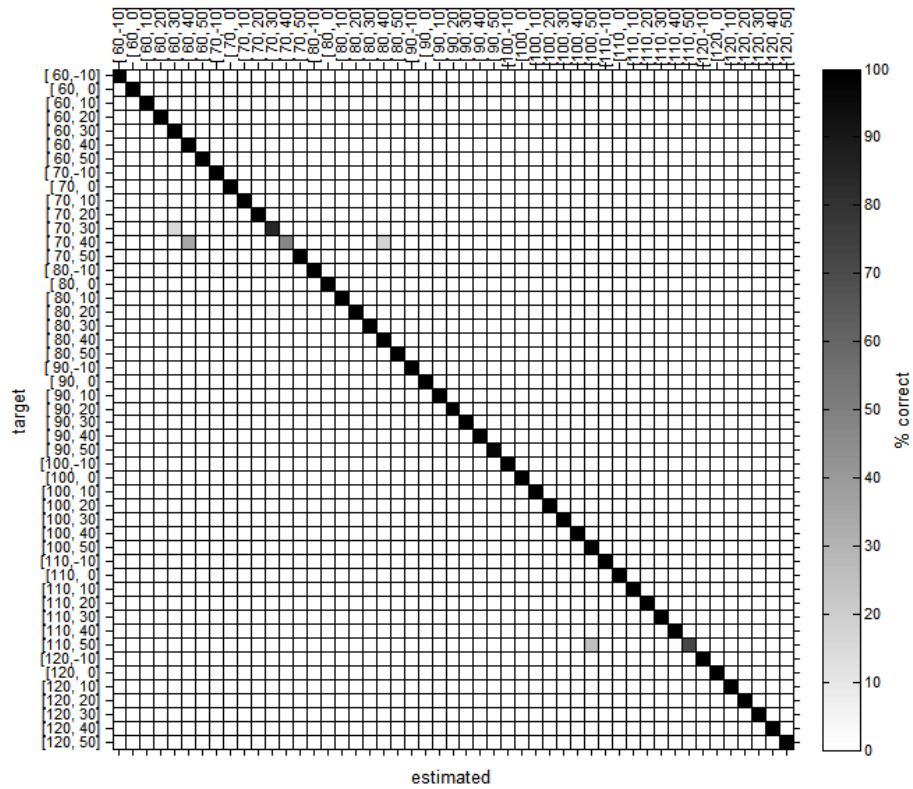


(c) Sprache; Vortraining mit Bottleneck; seitlich; parallel; ITD



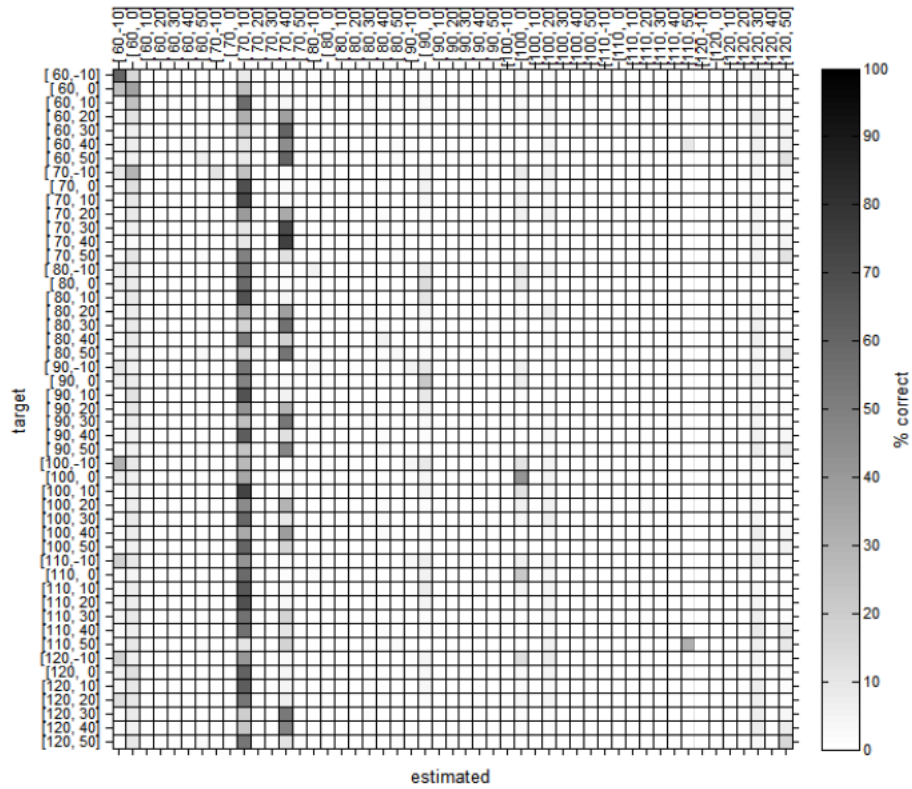
(d) Sprache; Vortraining mit Bottleneck; seitlich; parallel; Betragsspektrum



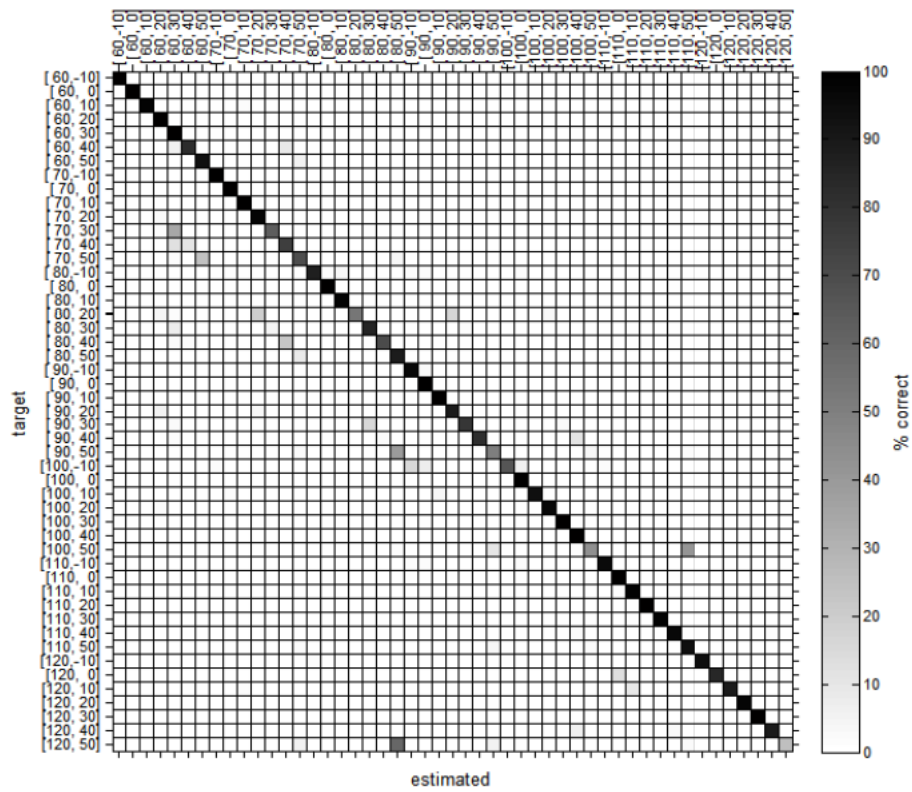


(e) Sprache; Vortraining mit Bottleneck; seitlich; parallel; **mfcc**

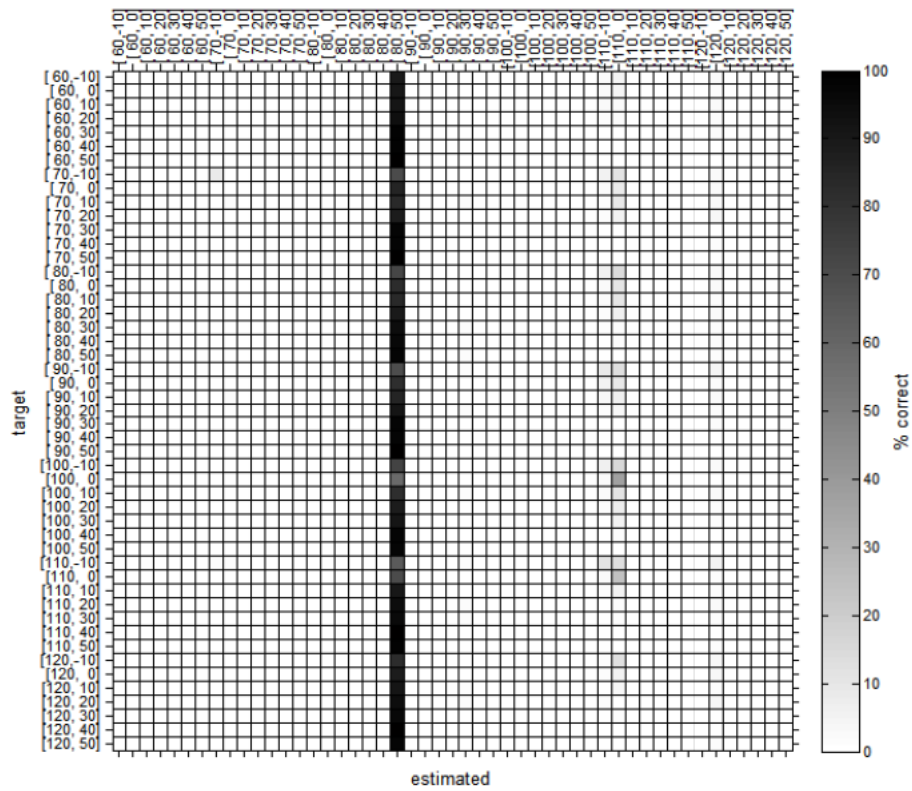
Abbildung A.17: Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit Bottleneck; seitlich; angehängt;



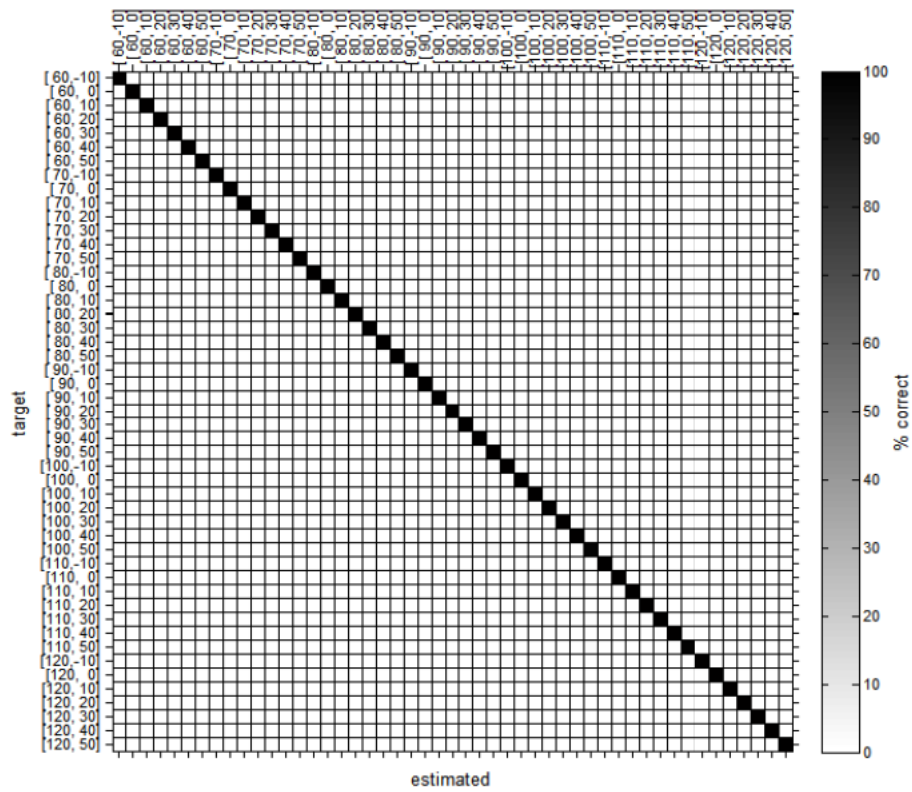
(a) Sprache; Vortraining mit Bottleneck; seitlich; sphärisch; ILD-ITD



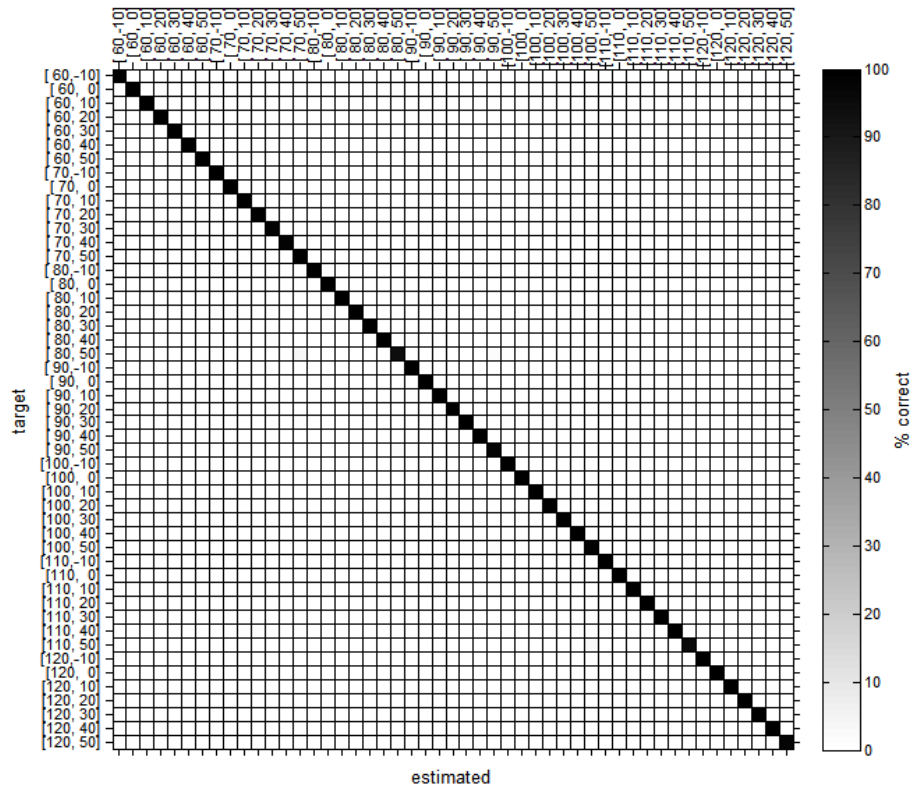
(b) Sprache; Vortraining mit Bottleneck; seitlich; sphärisch; ILD



(c) Sprache; Vortraining mit Bottleneck; seitlich; sphärisch; ITD

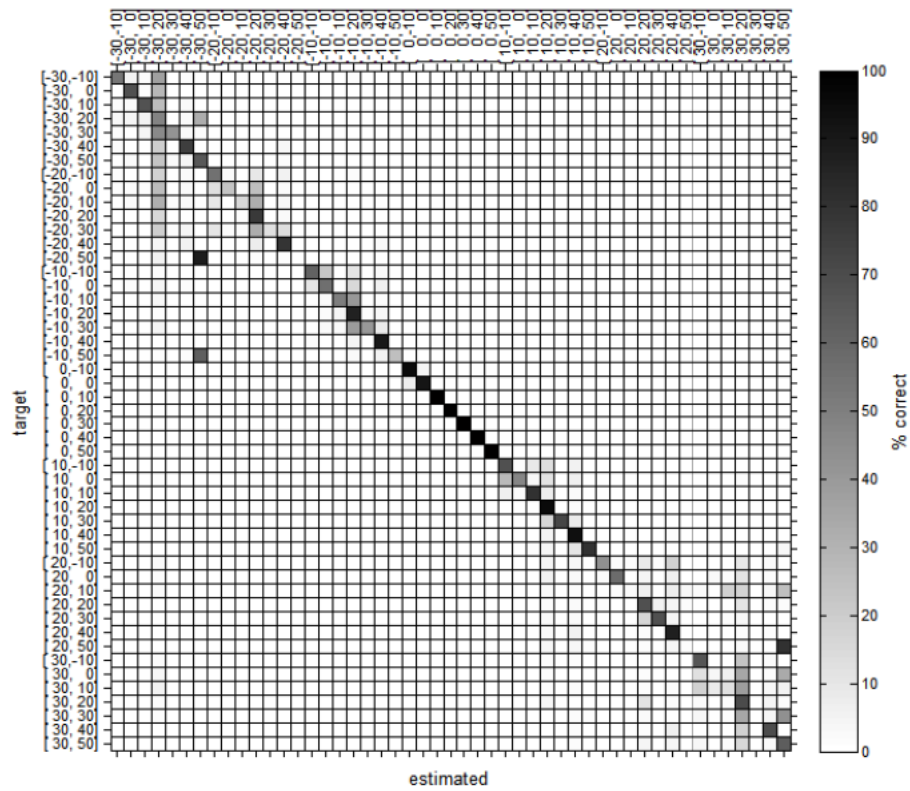


(d) Sprache; Vortraining mit Bottleneck; seitlich; sphärisch; Betragsspektrum

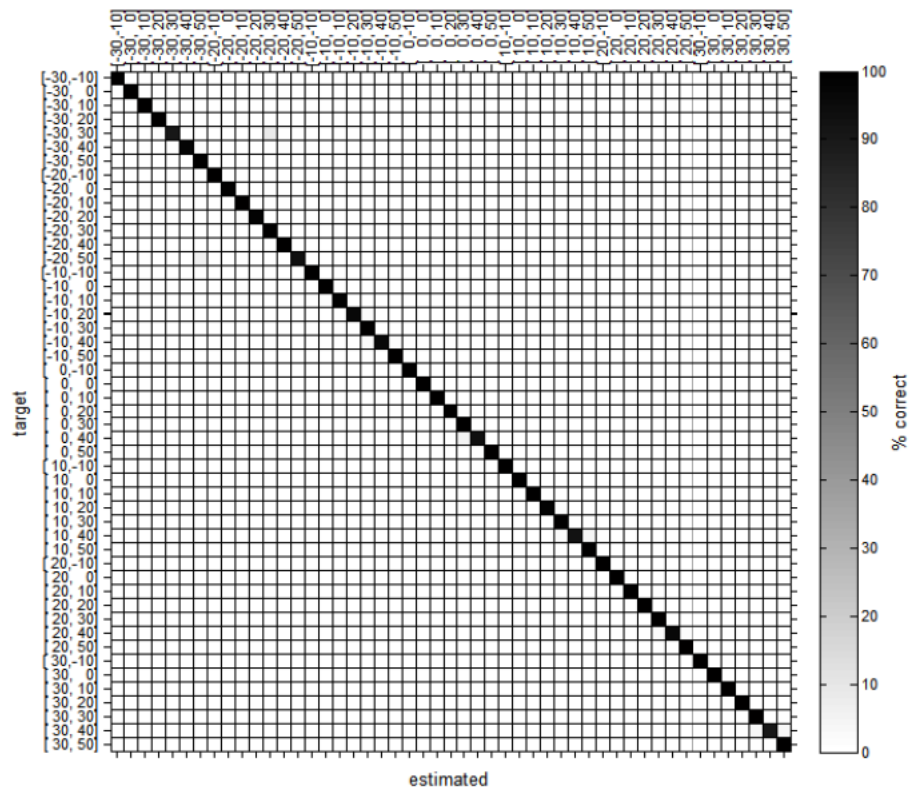


(e) Sprache; Vortraining mit Bottleneck; seitlich; sphärisch; **mfcc**

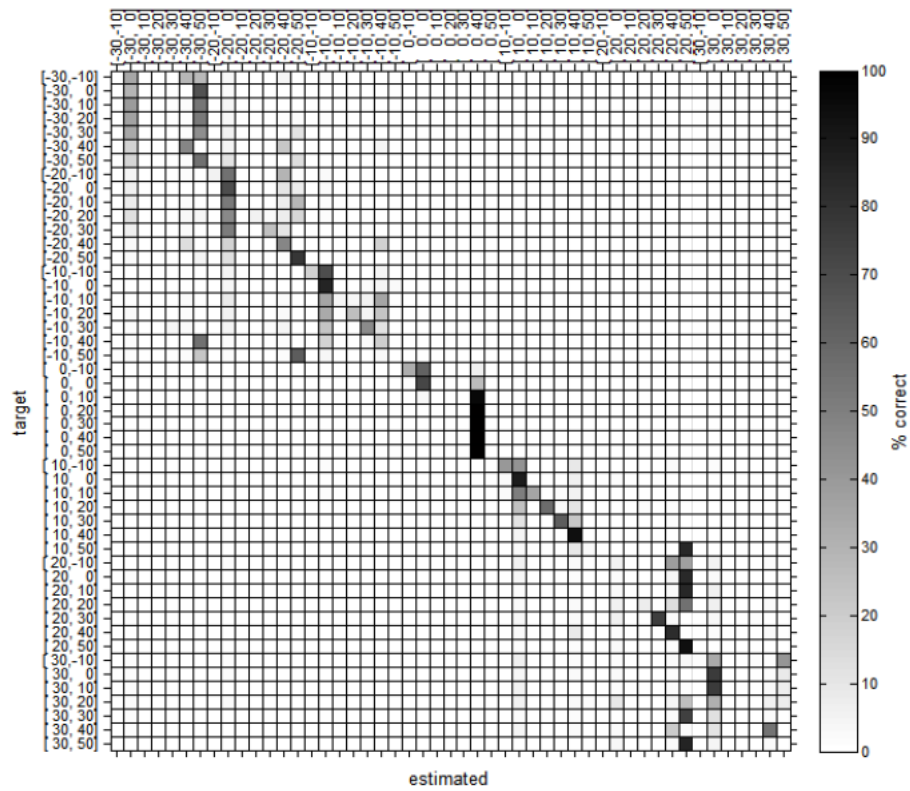
Abbildung A.18: Confusion-Matrizen für die Konditionen: Sprache; Vortraining mit Bottleneck; seitlich; sphärisch;



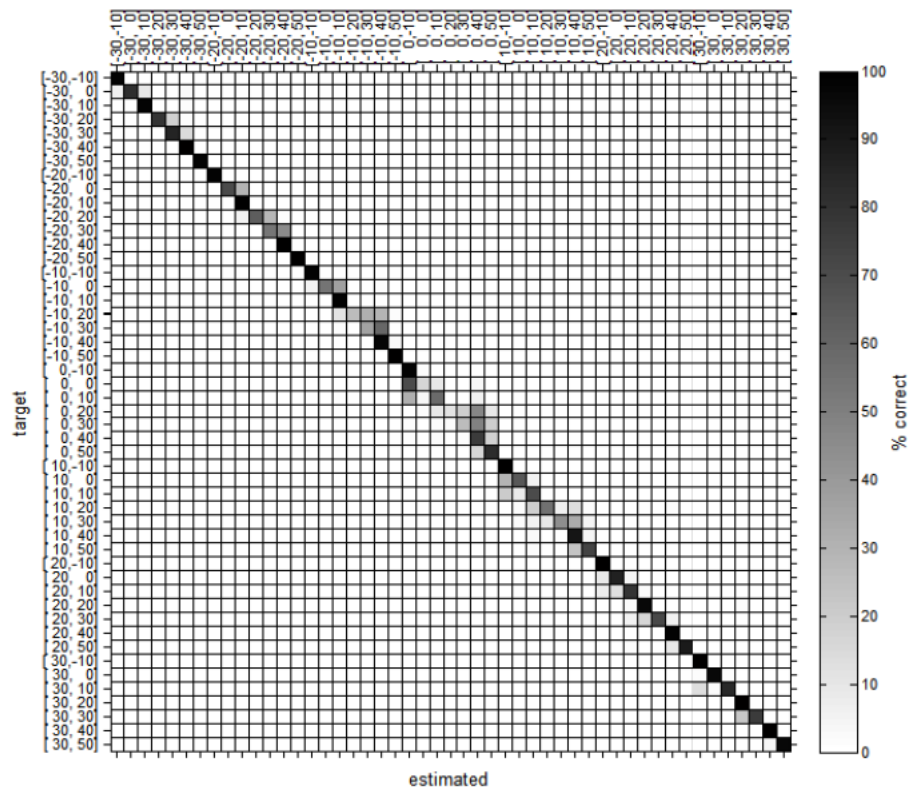
(a) Sprache; ohne Bottleneck-Vortraining; frontal; parallel; ILD-ITD



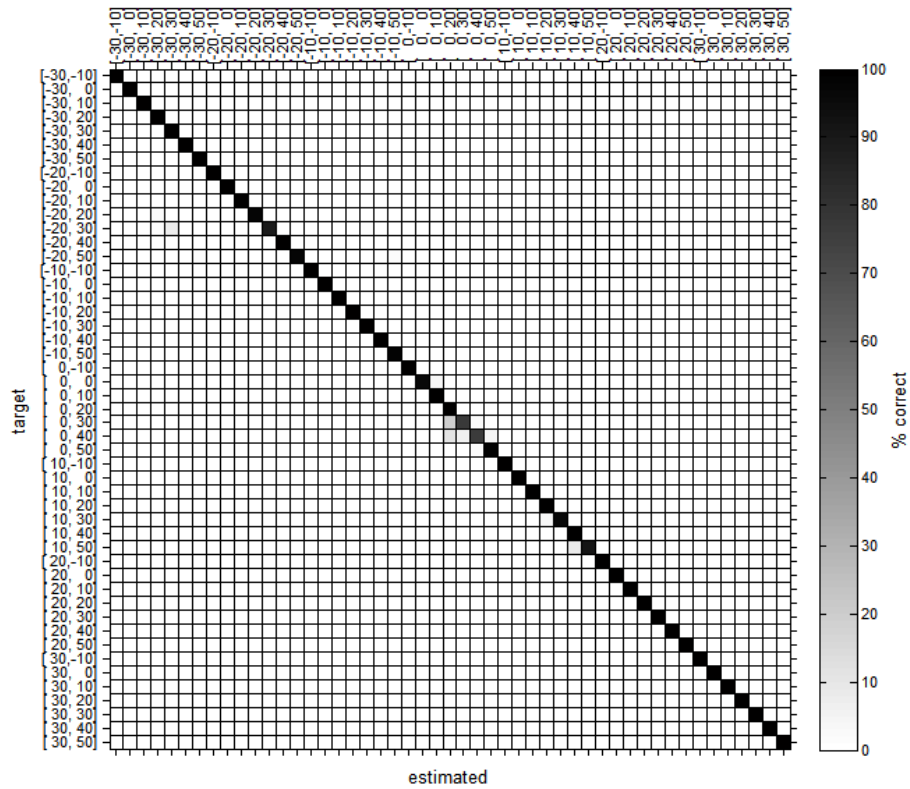
(b) Sprache; ohne Bottleneck-Vortraining; frontal; parallel; ILD



(c) Sprache; ohne Bottleneck-Vortraining; frontal; parallel; ITD



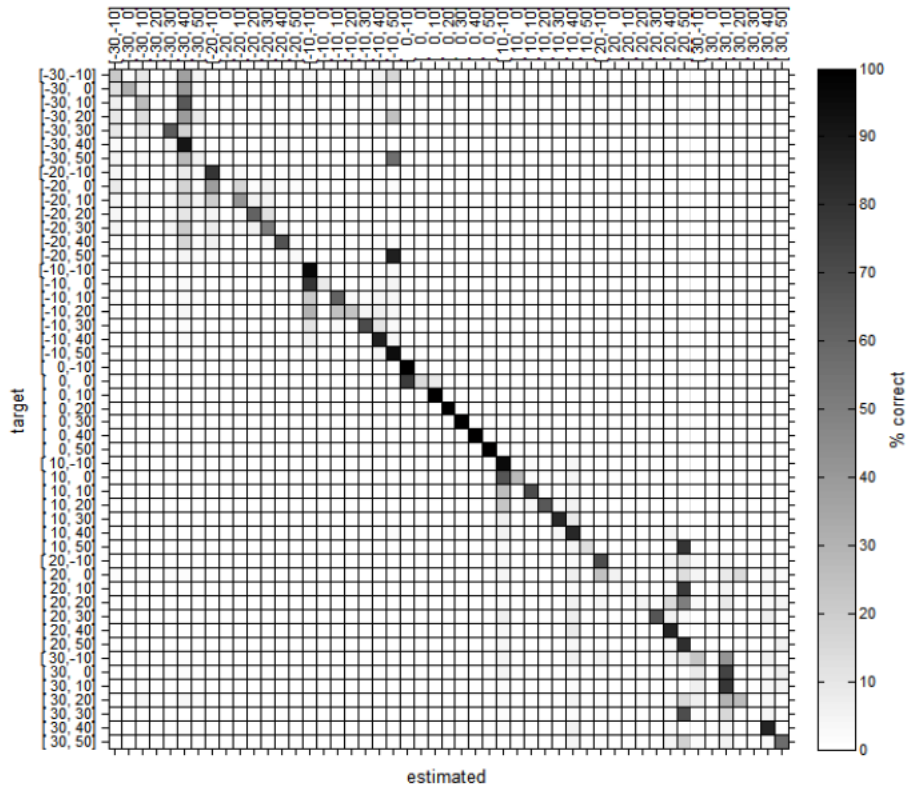
(d) Sprache; ohne Bottleneck-Vortraining; frontal; parallel; Betragsspektrum



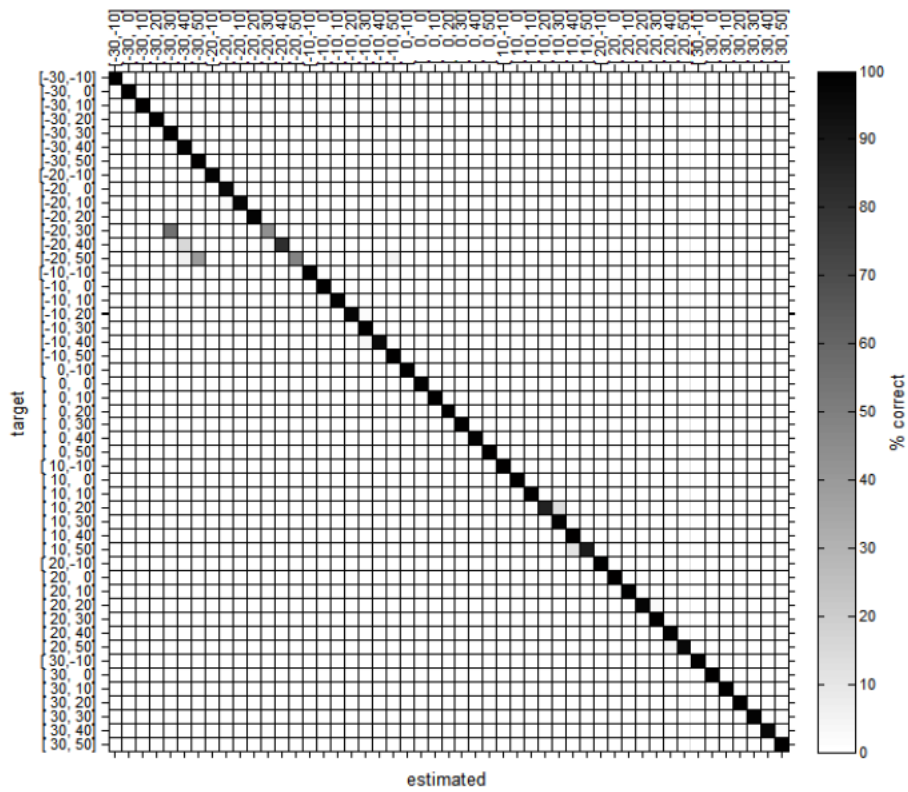
(e) Sprache; ohne Bottleneck-Vortraining; frontal; parallel; **mfcc**

Abbildung A.19: Confusion-Matrizen für die Konditionen: Sprache; ohne Bottleneck-Vortraining; frontal; parallel;



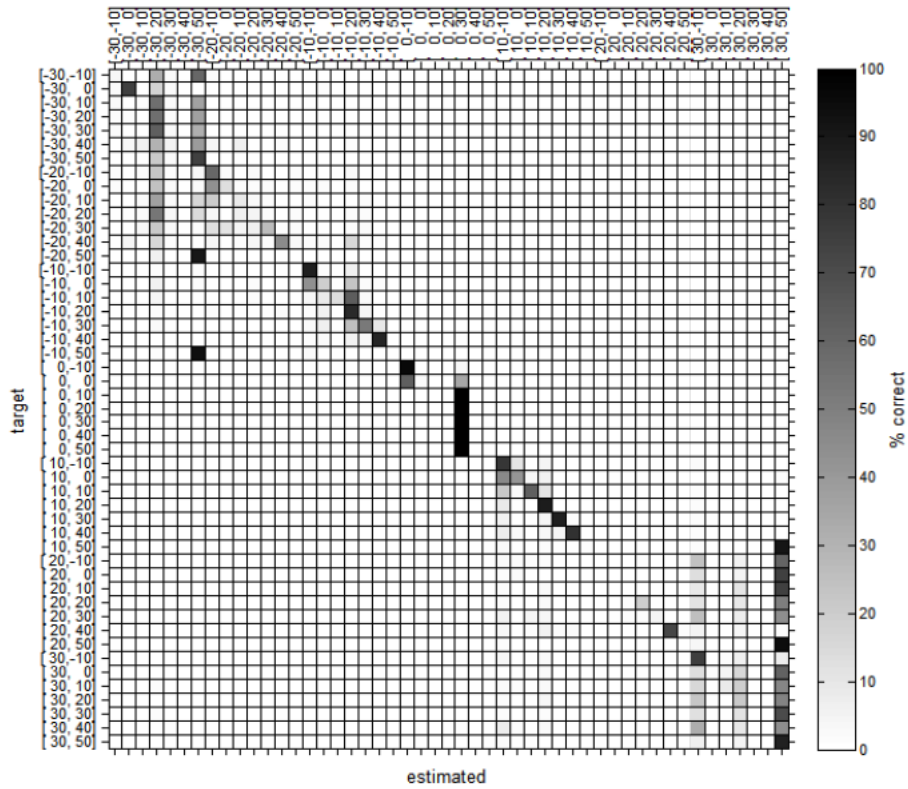


(a) Sprache; ohne Bottleneck-Vortraining; frontal; angehängt; ILD-ITD

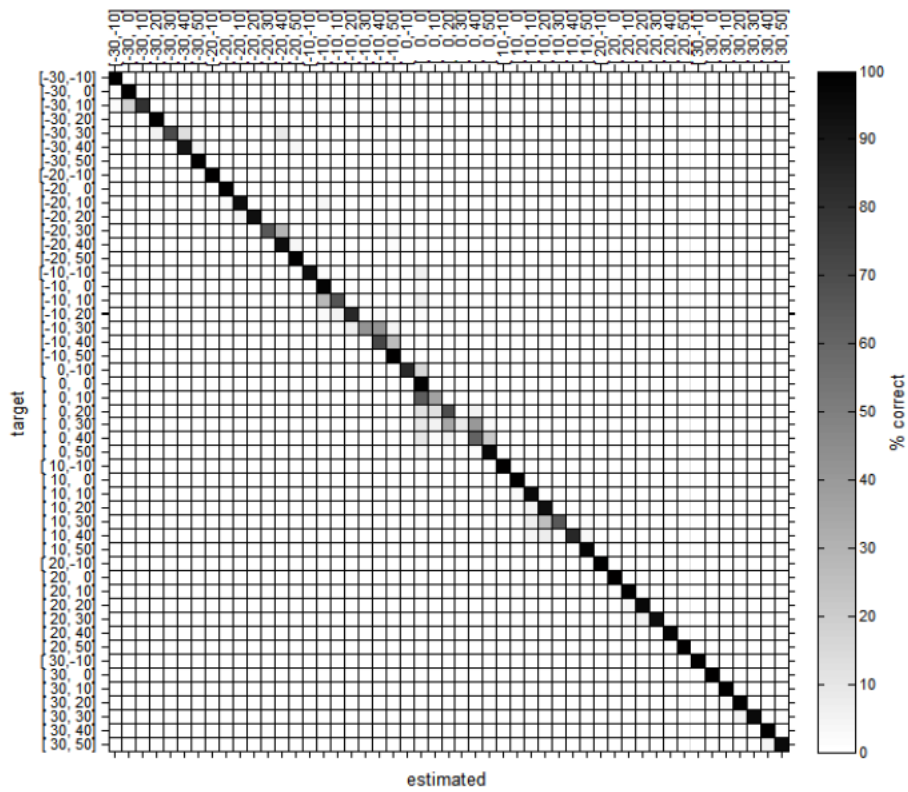


(b) Sprache; ohne Bottleneck-Vortraining; frontal; angehängt; ILD

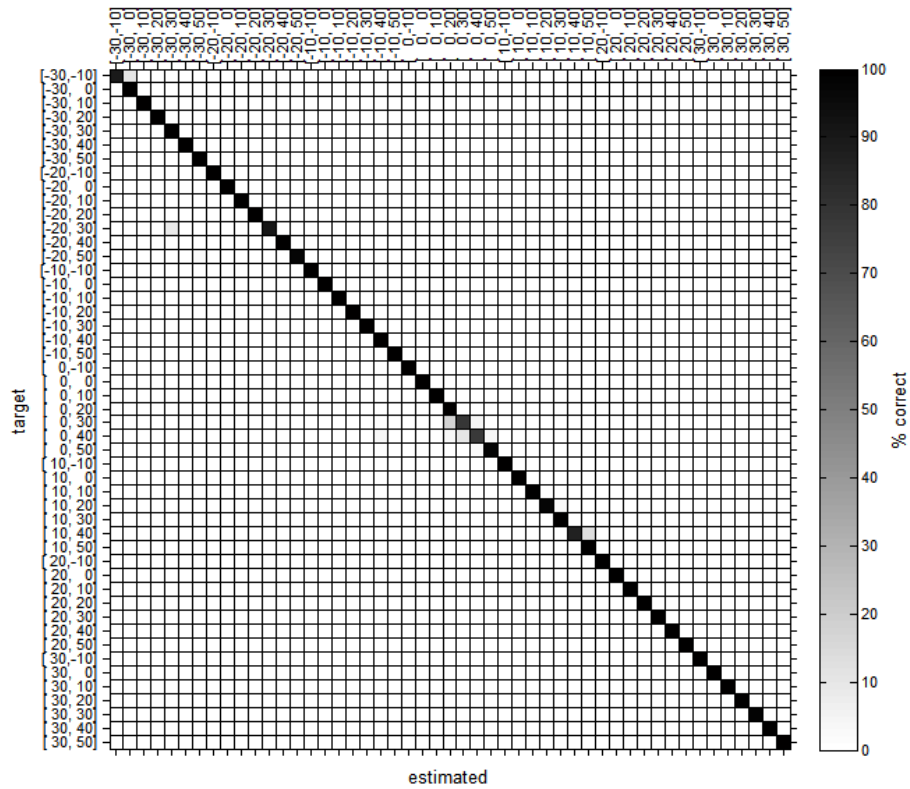




(c) Sprache; ohne Bottleneck-Vortraining; frontal; angehängt; ITD

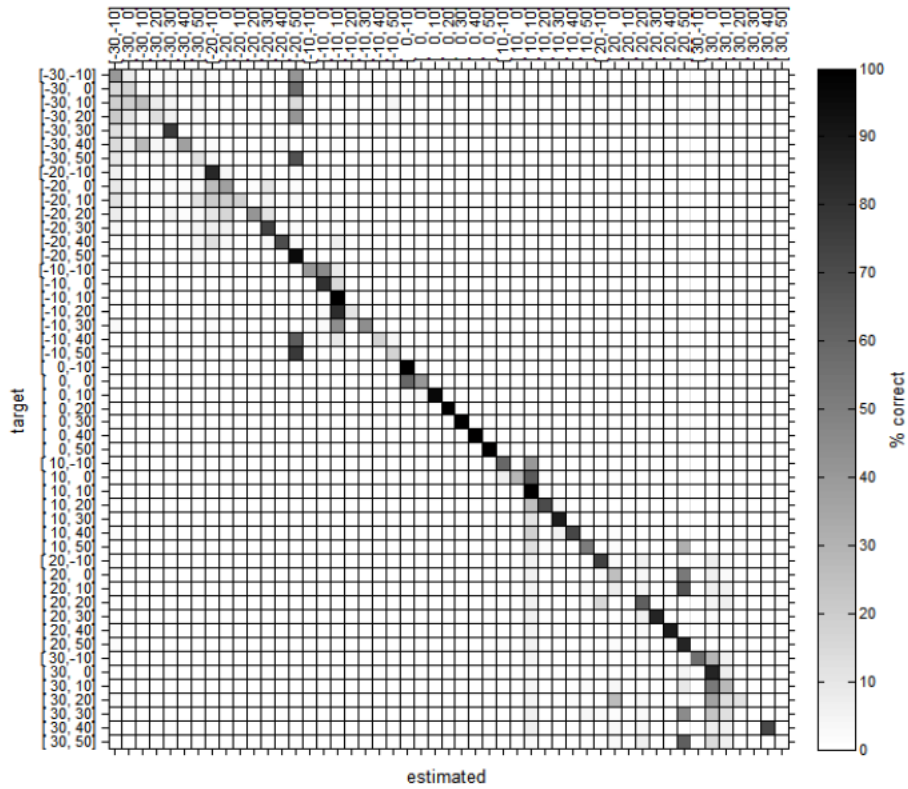


(d) Sprache; ohne Bottleneck-Vortraining; frontal; angehängt; Betragsspektrum

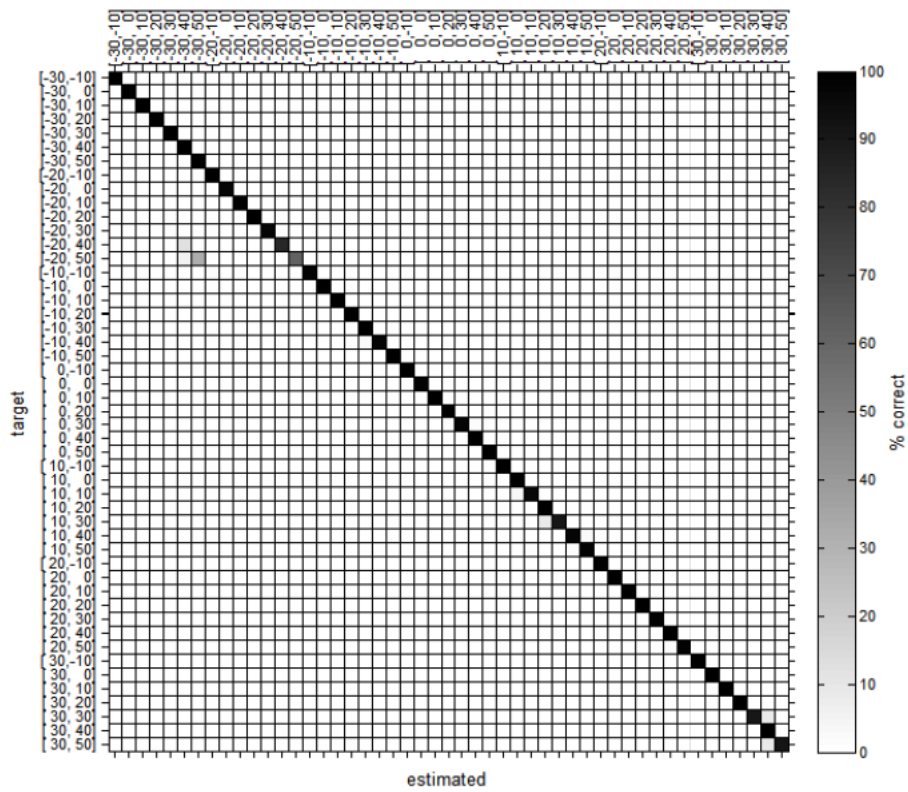


(e) Sprache; ohne Bottleneck-Vortraining; frontal; angehängt; **mfcc**

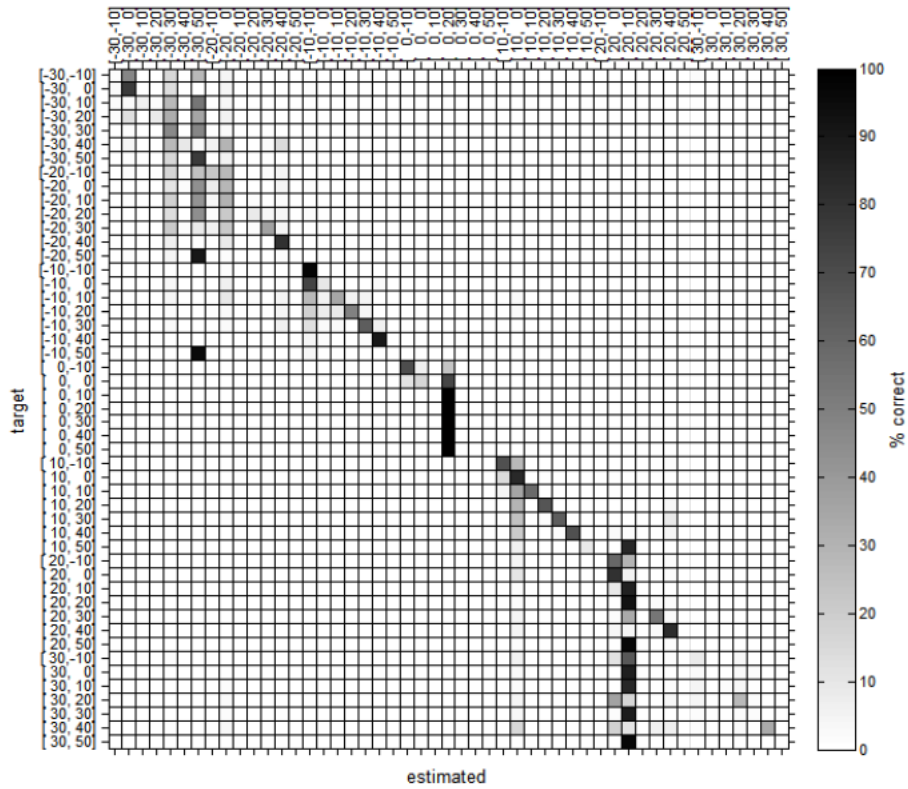
Abbildung A.20: Confusion-Matrizen für die Konditionen: Sprache; ohne Bottleneck-Vortraining; frontal; angehängt;



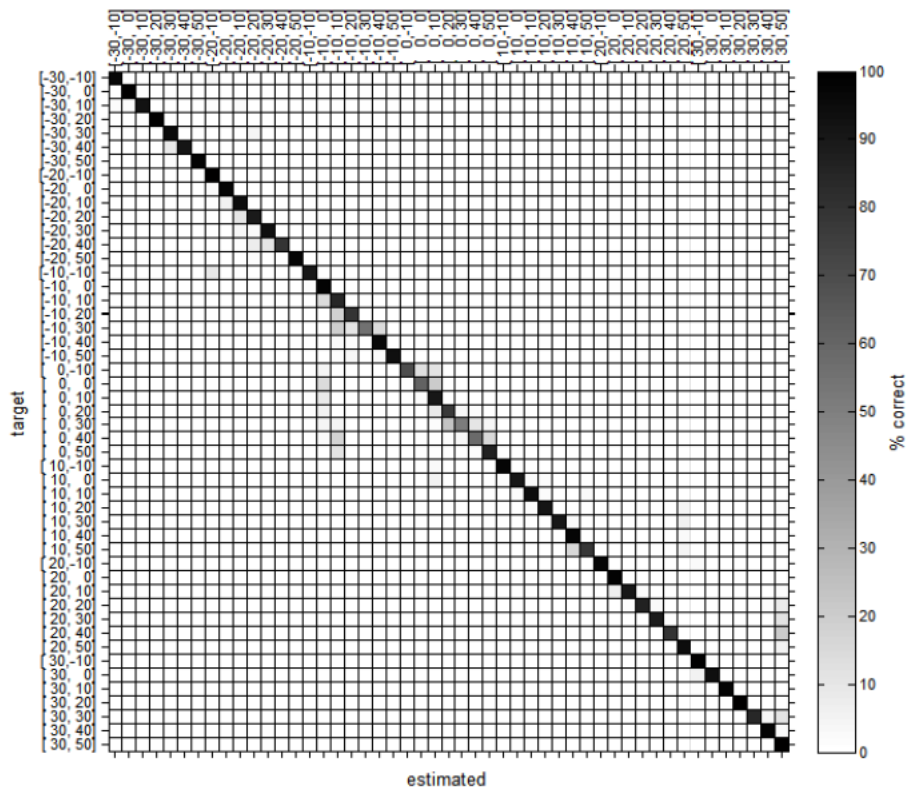
(a) Sprache; ohne Bottleneck-Vortraining; frontal; sphärisch; ILD-ITD



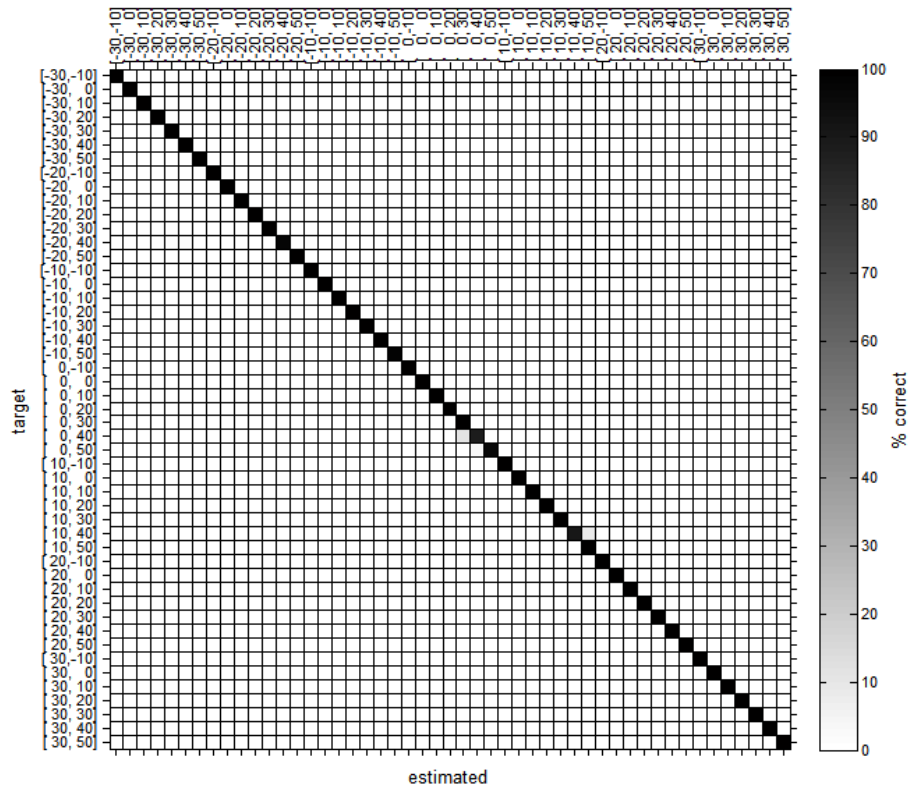
(b) Sprache; ohne Bottleneck-Vortraining; frontal; sphärisch; ILD



(c) Sprache; ohne Bottleneck-Vortraining; frontal; sphärisch; ITD

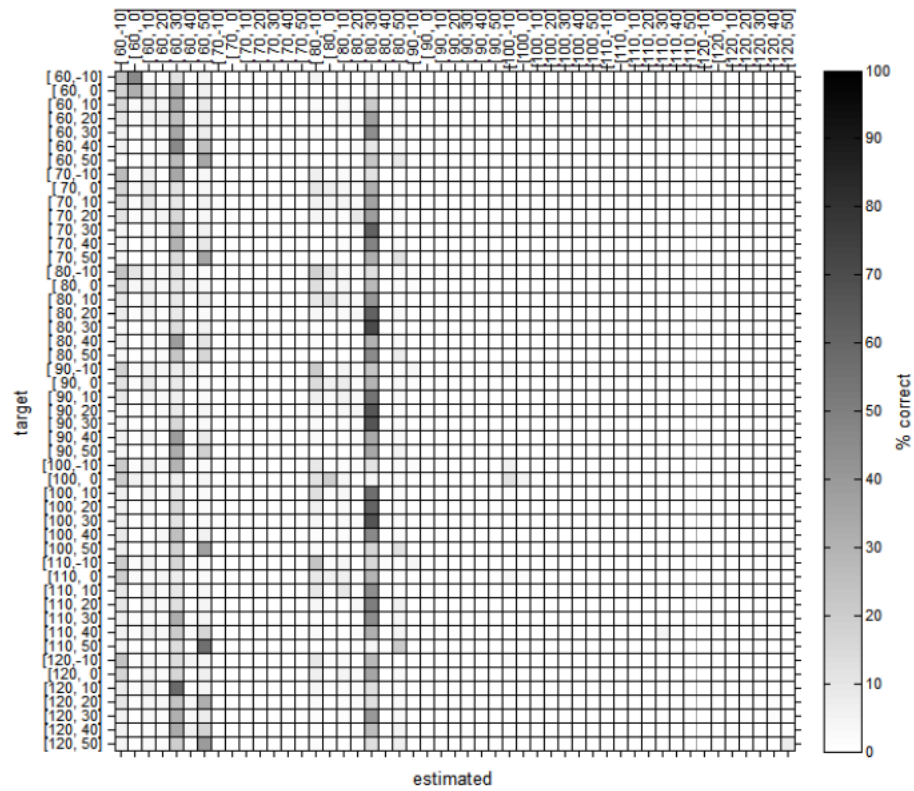


(d) Sprache; ohne Bottleneck-Vortraining; frontal; sphärisch; Betragsspektrum

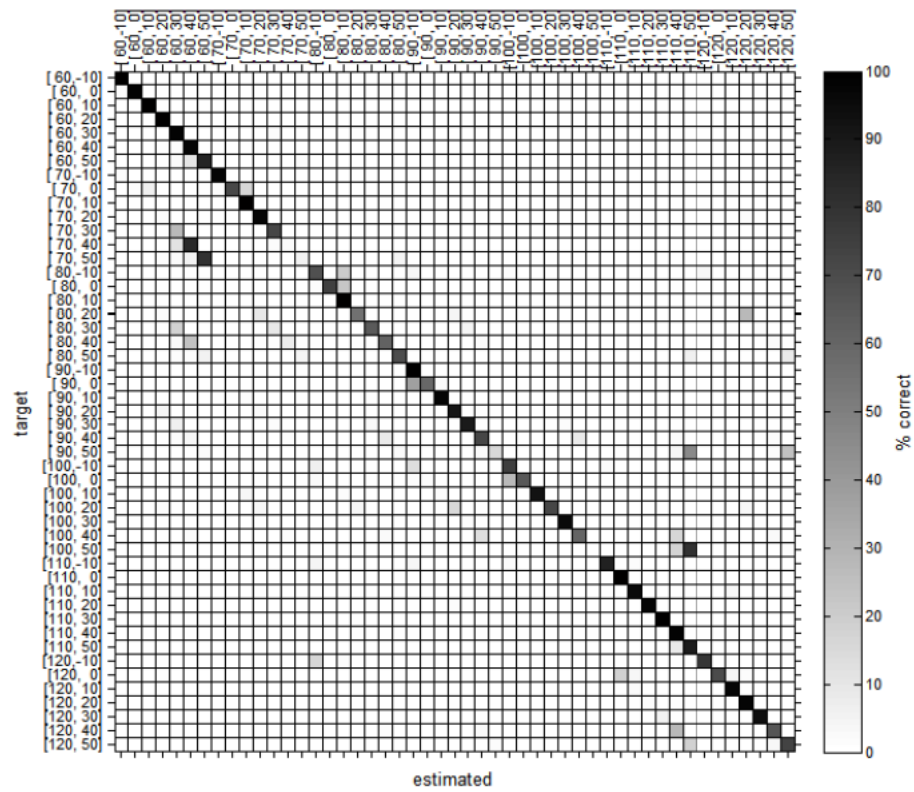


(e) Sprache; ohne Bottleneck-Vortraining; frontal; sphärisch; **mfcc**

Abbildung A.21: Confusion-Matrizen für die Konditionen: Sprache; ohne Bottleneck-Vortraining; frontal; sphärisch;

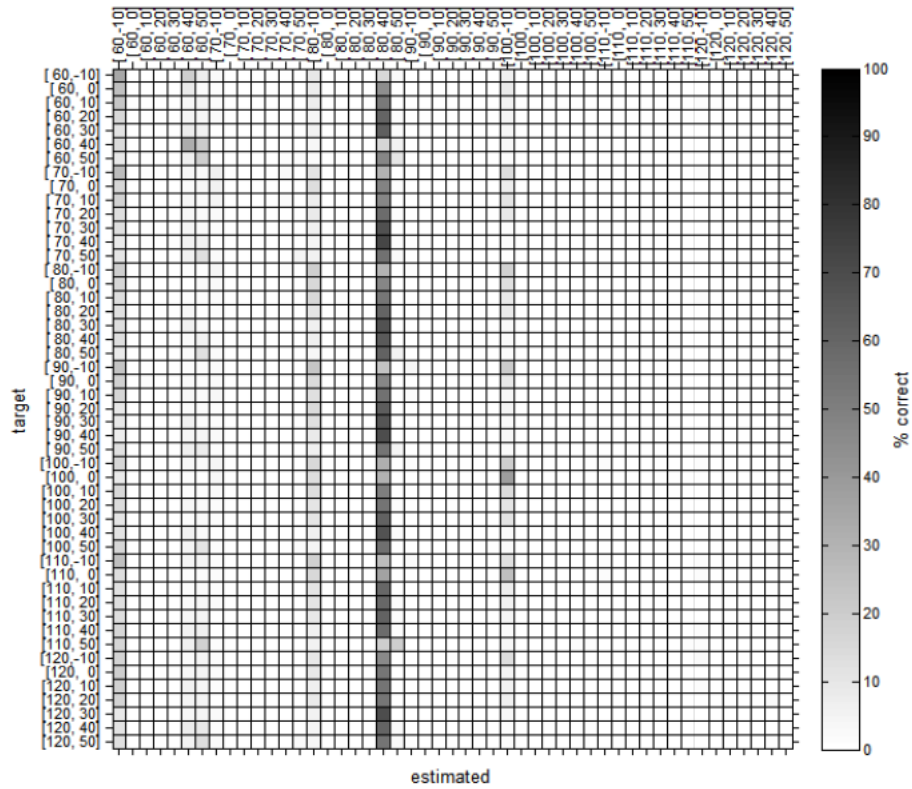


(a) Sprache; ohne Bottleneck-Vortraining; seitlich; parallel; ILD-ITD

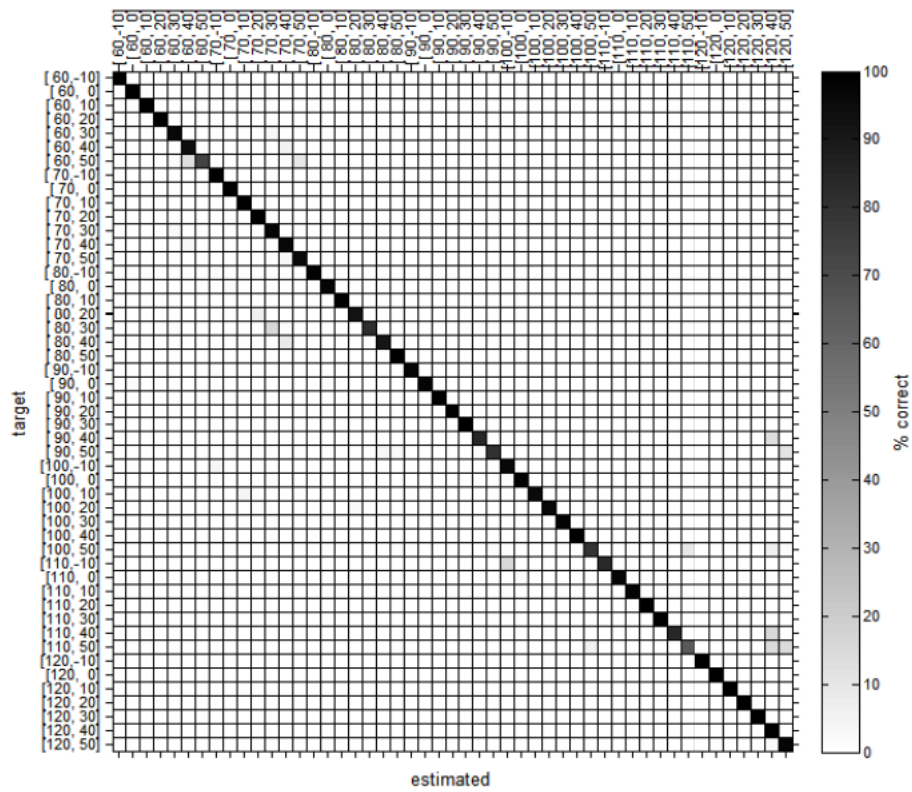


(b) Sprache; ohne Bottleneck-Vortraining; seitlich; parallel; ILD

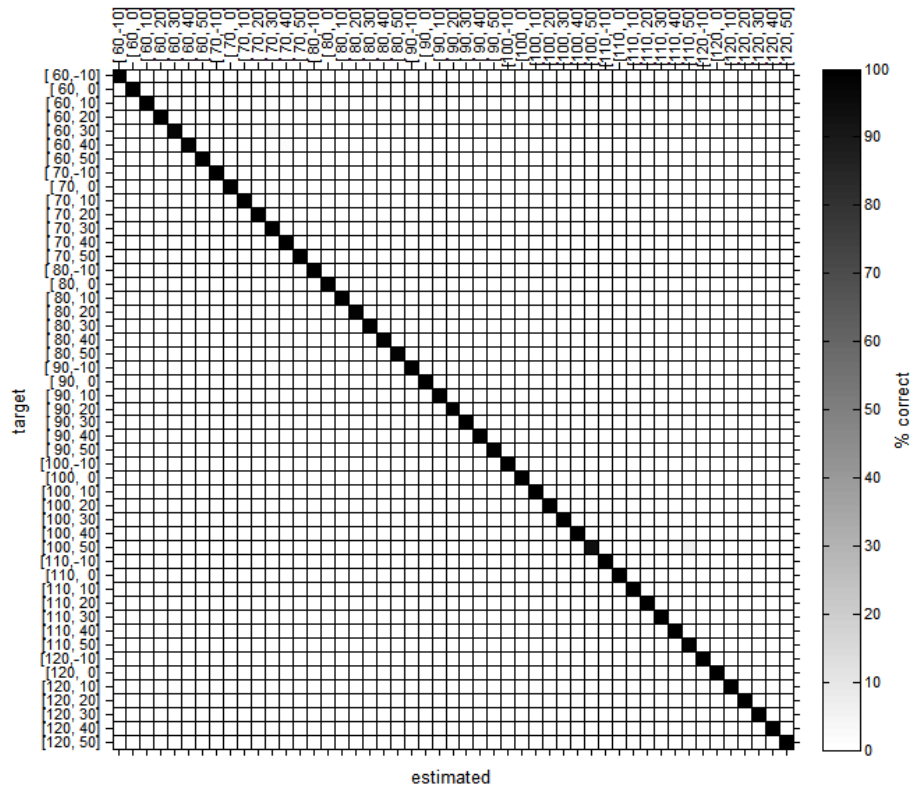




(c) Sprache; ohne Bottleneck-Vortraining; seitlich; parallel; ITD



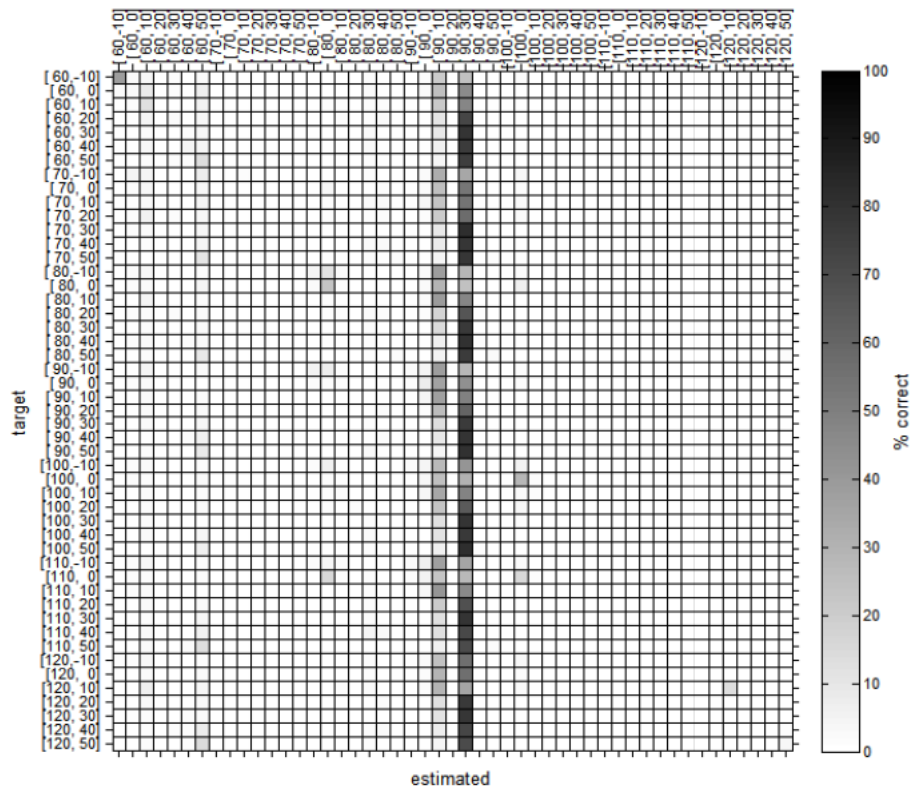
(d) Sprache; ohne Bottleneck-Vortraining; seitlich; parallel; Betragsspektrum



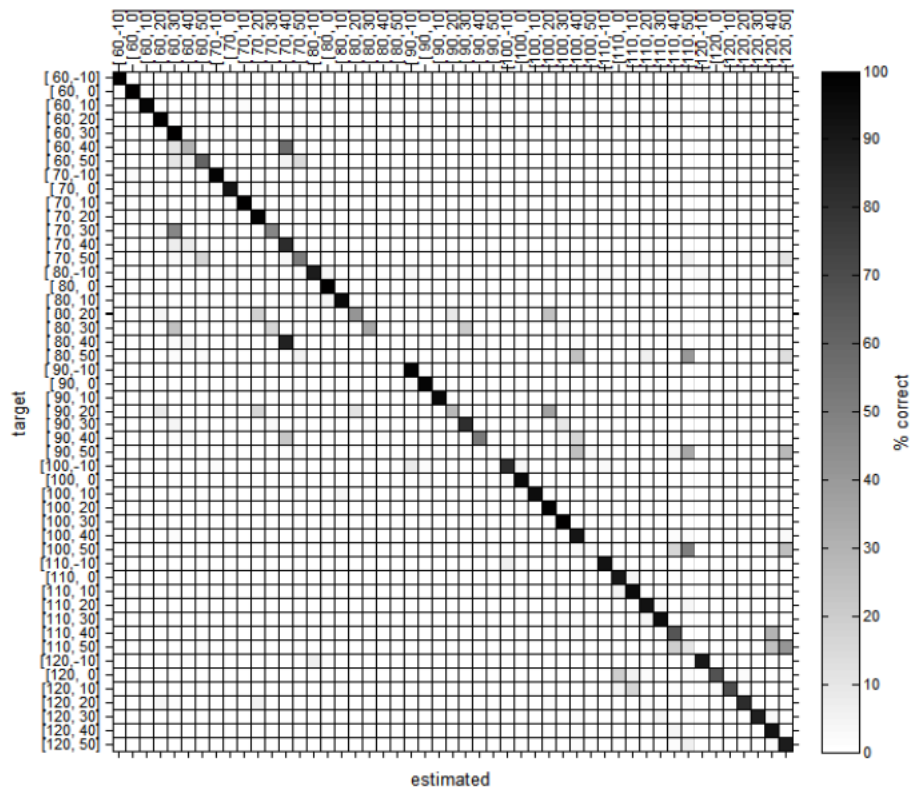
(e) Sprache; ohne Bottleneck-Vortraining; seitlich; parallel; **mfcc**

Abbildung A.22: Confusion-Matrizen für die Konditionen: Sprache; ohne Bottleneck-Vortraining; seitlich; parallel;

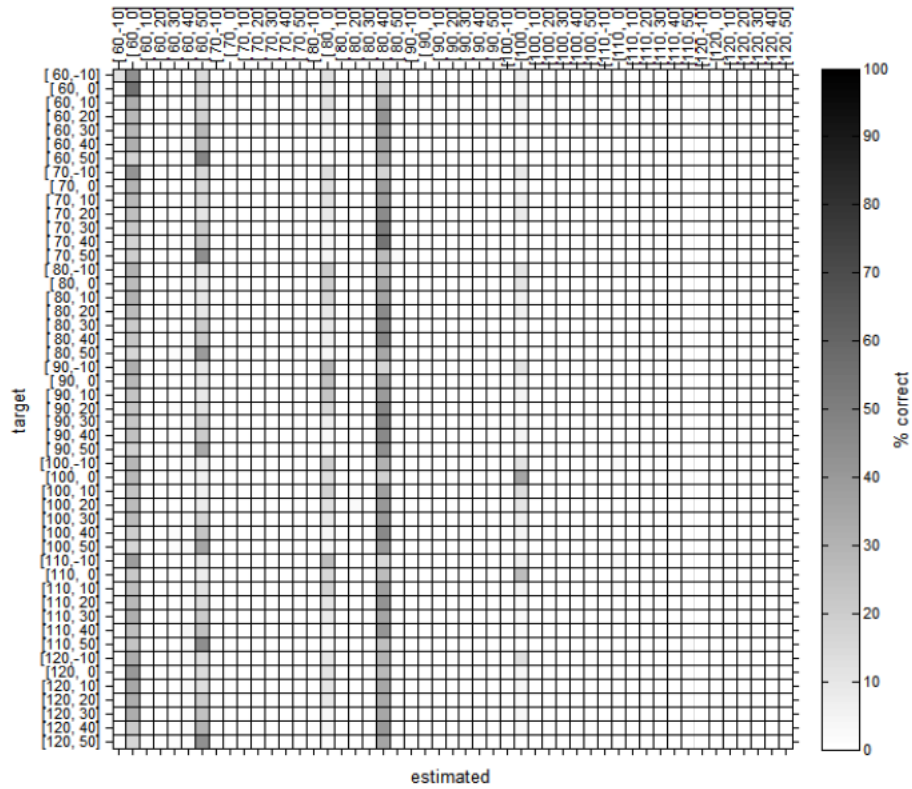




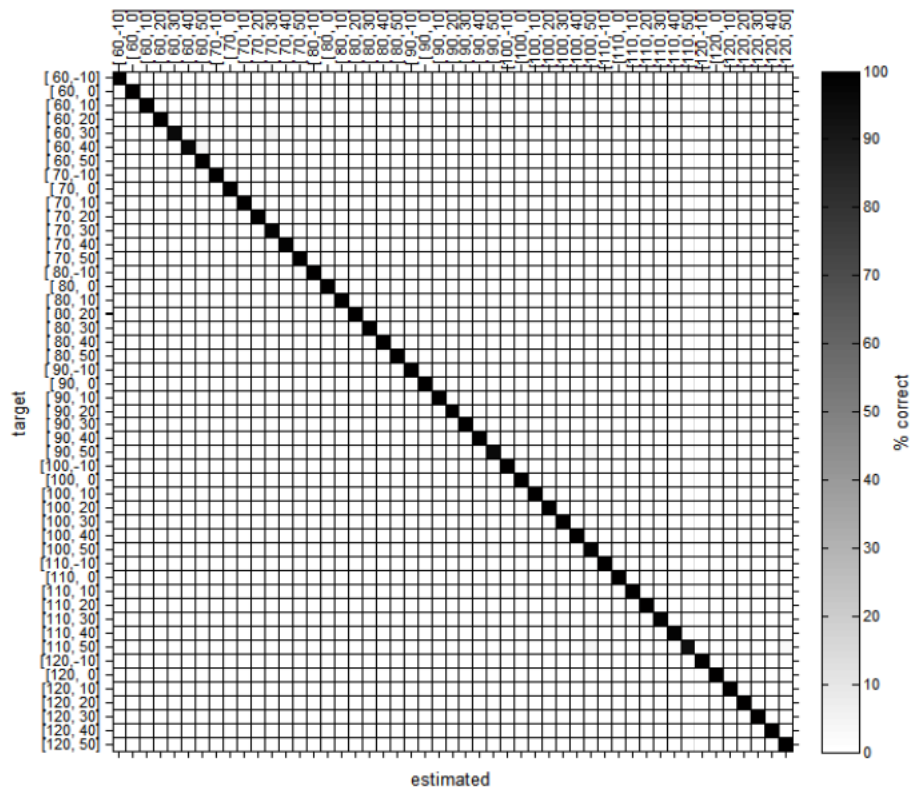
(a) Sprache; ohne Bottleneck-Vortraining; seitlich; angehängt; ILD-ITD



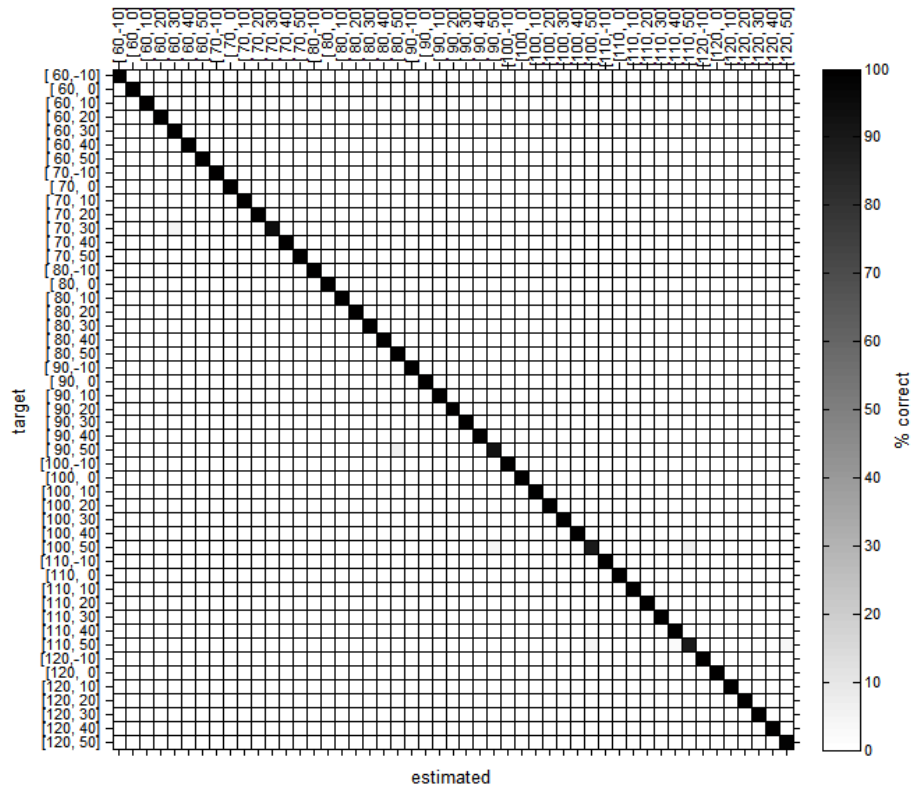
(b) Sprache; ohne Bottleneck-Vortraining; seitlich; angehängt; ILD



(c) Sprache; ohne Bottleneck-Vortraining; seitlich; angehängt; ITD

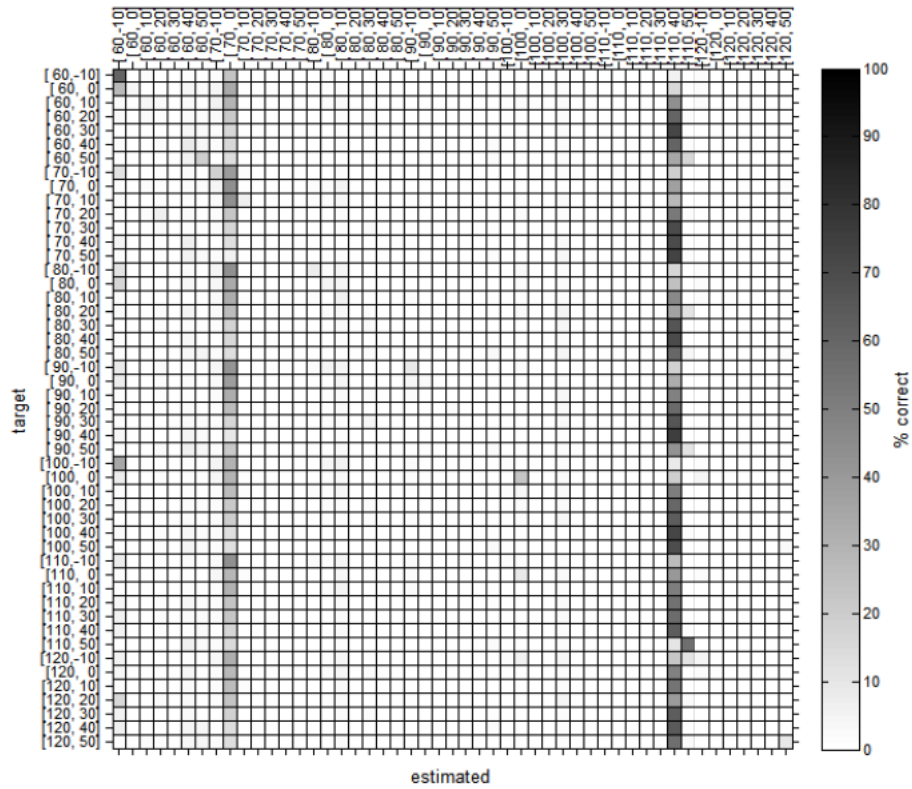


(d) Sprache; ohne Bottleneck-Vortraining; seitlich; angehängt; Betragsspektrum

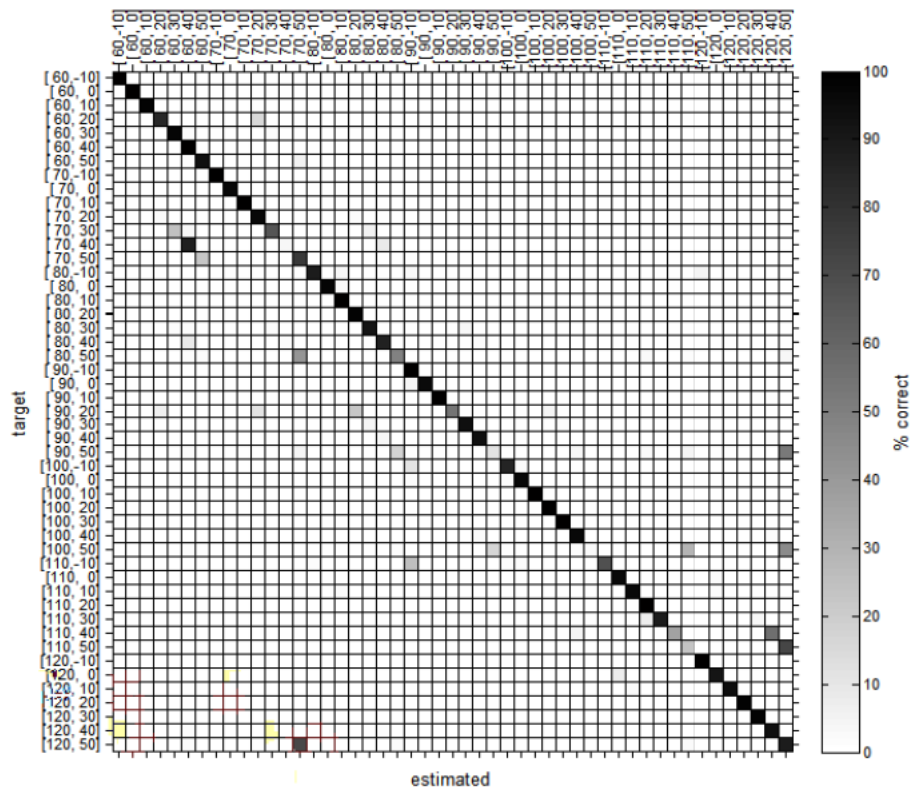


(e) Sprache; ohne Bottleneck-Vortraining; seitlich; angehängt; **mfcc**

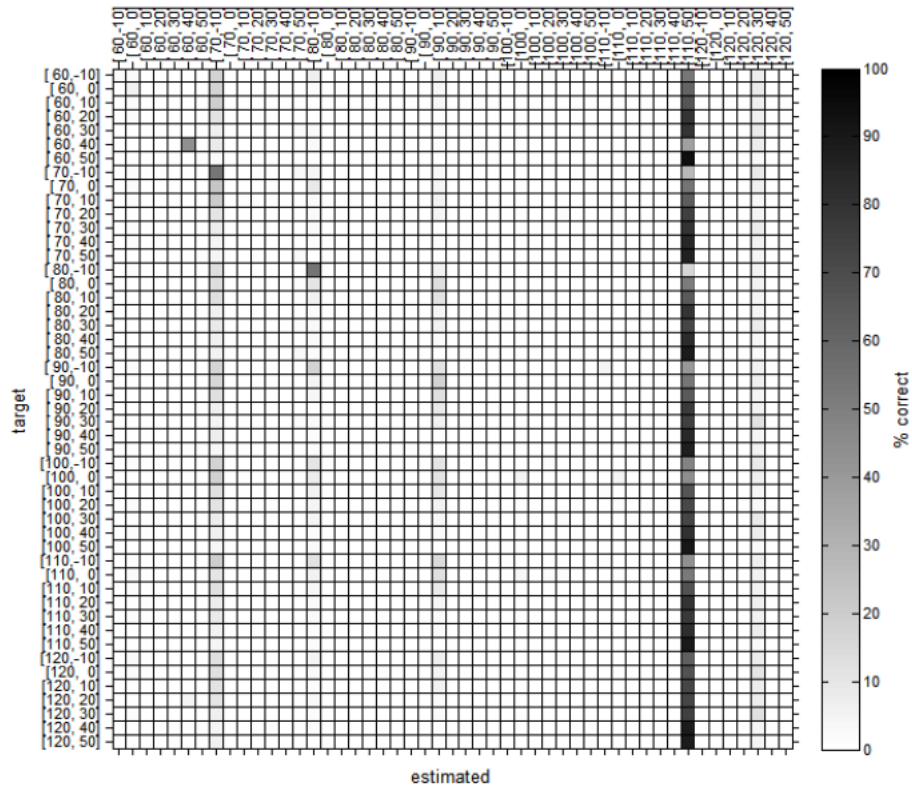
Abbildung A.23: Confusion-Matrizen für die Konditionen: Sprache; ohne Bottleneck-Vortraining; seitlich; angehängt;



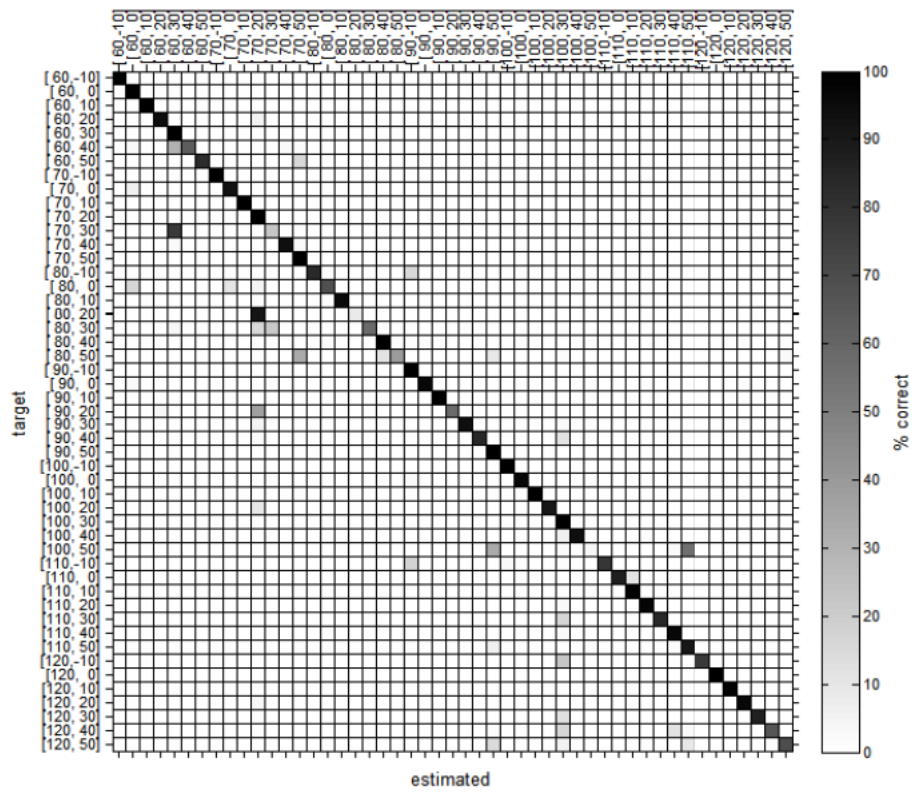
(a) Sprache; ohne Bottleneck-Vortraining; seitlich; sphärisch; ILD-ITD



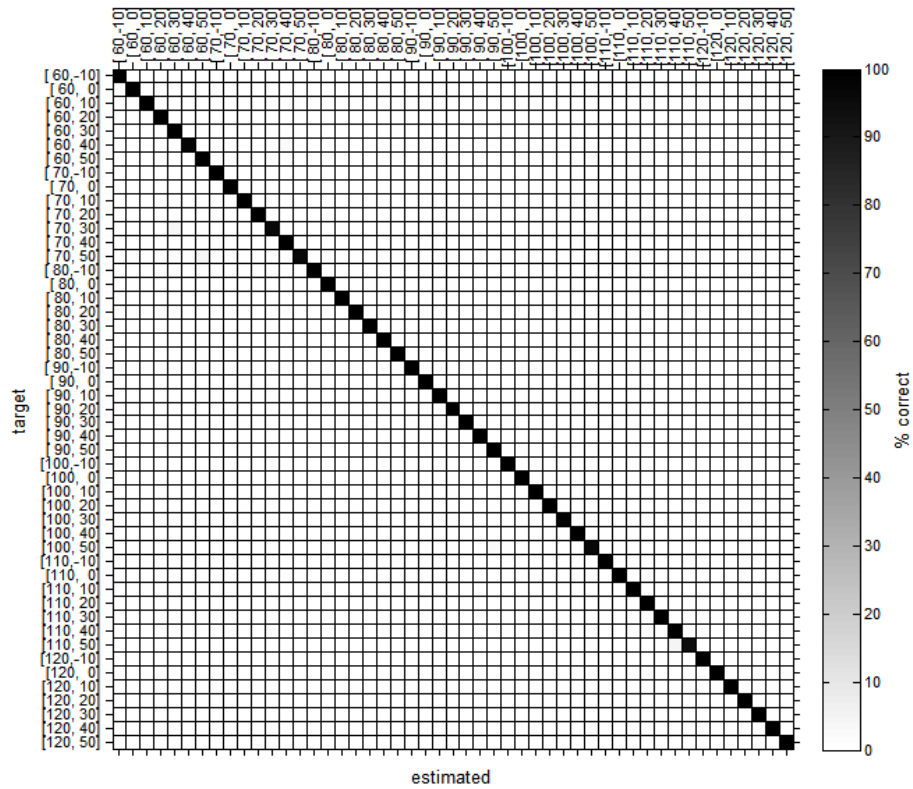
(b) Sprache; ohne Bottleneck-Vortraining; seitlich; sphärisch; ILD



(c) Sprache; ohne Bottleneck-Vortraining; seitlich; sphärisch; ITD



(d) Sprache; ohne Bottleneck-Vortraining; seitlich; sphärisch; Betragsspektrum



(e) Sprache; ohne Bottleneck-Vortraining; seitlich; sphärisch; **mfcc**

Abbildung A.24: Confusion-Matrizen für die Konditionen: Sprache; ohne Bottleneck-Vortraining; seitlich; sphärisch;

Anhang B

Tabellenanhang

Anhang C

## Quellcode Matlab

i HRTF Processing



```
% processing dataset
% 1.) seperate hrirs for speaker 1 and 2
% 2.) shorten ir to 425 samples
% 3.) sin/cos-window
clear; close all; clc;

%
hor=0;
med=0;
combined=1;

mode= 'front'; %'full' 'side' 'front'

%
if strcmp(mode, 'front')
    grid_az=[0:10:30 330:10:358]; % highest resolution/available HRTF 0:2:358 !!!
    grid_el=-10:10:50; % highest resolution/available HRTF -10:2:68 !!!
    plot=0;
elseif strcmp(mode, 'side')
    grid_az=60:10:120; % highest resolution/available HRTF 0:2:358 !!!
    grid_el=-10:10:50; % highest resolution/available HRTF -10:2:68 !!!
    plot=0;
elseif strcmp(mode, 'full')
    grid_az=0:2:358; % highest resolution/available HRTF 0:2:358 !!!
    grid_el=-10:2:68; % highest resolution/available HRTF -10:2:68 !!!

    if combined==1
        plot=1; % plot usefull for combined mode
    else
        plot=0;
    end
else
    error('choose mode: front, side or full')
end

%%

full_az=0:2:358;
full_el=-10:2:68;
%

del_az=full_az(find(ismember(full_az,grid_az)==0));
del_el=full_el(find(ismember(full_el,grid_el)==0));

%
%set(0,'defaulttextinterpreter','latex')

%%
N_old=425;
N=ceil(N_old/44100*16000);
N_fade=10;
hrir_window = ones(N,1);
```

```
hrir_window(1:N_fade+1) = sin(linspace(0,pi/2, N_fade+1)).^2; % noch mal in doku✓  
schauen ob nicht gefenstert, sonst eigentlich egal, da eh nullen am anfang  
hrir_window(end-N_fade:end) = cos(linspace(0,pi/2, N_fade+1)).^2;
```

```
HRIR_path=fullfile('HRTF');
```

```
files_HRIR = dir(HRIR_path);  
names_HRIR = {files_HRIR.name}; names_HRIR(1:2) = [];
```

```
len_names=length(names_HRIR);  
for kk=1:len_names  
    disp(names_HRIR{1,kk});  
  
    save_path=[HRIR_path '_short/' names_HRIR{kk}(1:end-4)];  
if ~exist(save_path, 'dir')  
    mkdir(save_path)  
end
```

```
load(fullfile(HRIR_path, names_HRIR{1,kk}));
```

```
if size(fulldb(1).ir,2)==8  
hrir_window_mat= repmat(hrir_window, [1 8]);  
elseif size(fulldb(1).ir,2)==2  
    hrir_window_mat= repmat(hrir_window, [1 2]);  
end
```

```
% 1.)  
az=[];  
el=[];  
speaker=[];  
for pp=1:length(fulldb)  
az(pp,1)=fulldb(1,pp).az;  
el(pp,1)=fulldb(1,pp).el;  
speaker(pp,1)=str2num(fulldb(1,pp).speaker(2));  
end
```

```
[AzElSpeaker ind]=sortrows([az el speaker],3);  
speaker=speaker(ind);  
az=az(ind); el=el(ind);  
fulldb=fulldb(ind);
```

```
[~, ix, dupl]=unique(AzElSpeaker(:,1:2), 'rows', 'first');
```

```
ind=1:length(az);  
ind(ix)=[];  
fulldb(ind)=[];
```

```
% delete elevation <-10  
az=[];  
el=[];
```

```
speaker=[];
for pp=1:length(fulldb)
az(pp,1)=fulldb(1,pp).az;
el(pp,1)=fulldb(1,pp).el;
speaker(pp,1)=str2num(fulldb(1,pp).speaker(2));
end

ind_el_neg=find(el<-10);
ind_el_pos=find(el>68);
ind_el=[ind_el_neg; ind_el_pos];
az(ind_el)=[];
el(ind_el)=[];
fulldb(ind_el)=[];

for pp=1:length(fulldb)
az(pp,1)=fulldb(1,pp).az;
el(pp,1)=fulldb(1,pp).el;
speaker(pp,1)=str2num(fulldb(1,pp).speaker(2));
end

% resolution to 10Â°
%ind_el=find(ismember(el,[-9:-1 1:9 11:19 21:29 31:39 41:49 51:68])==1);
%ind_el=[];%find(ismember(el,[-9:-1 1:9 11:19 21:29 31:39 41:49 51:68])==1);
%ind_az=[];%find(ismember(az,[2:4:358])==1);
% to 30Â° for az for frontal circle
%ind_az=find(ismember(az,[1:29 31:59 61:89 91:269 271:299 301:329 331:360])==1);
%ind_az=find(ismember(az,[1:9 11:19 21:29 31:329 331:339 341:349 351:360])==1);

ind_az=find(ismember(az,del_az)==1);
ind_el=find(ismember(el,del_el)==1);

ind_el_az=[ind_el; ind_az];
az(ind_el_az)=[];
el(ind_el_az)=[];
fulldb(ind_el_az)=[];

for pp=1:length(fulldb)
az(pp,1)=fulldb(1,pp).az;
el(pp,1)=fulldb(1,pp).el;
speaker(pp,1)=str2num(fulldb(1,pp).speaker(2));
end

[~,ind_sort]=sortrows([el az]);
az=az(ind_sort);
el=el(ind_sort);
fulldb=fulldb(ind_sort);
speaker=speaker(ind_sort);

% just horizontal plane
if hor==1
ind_el=find(el==0);
```

```
az=az(ind_el);
el=el(ind_el);
fulldb=fulldb(ind_el);

for pp=1:length(fulldb)
    %conversion of grid left ear -90, in front 0, right 90°
    if fulldb(1,pp).az>180
        fulldb(1,pp).az=fulldb(1,pp).az-360;
    end
    az(pp,1)=fulldb(1,pp).az;
    el(pp,1)=fulldb(1,pp).el;
    speaker(pp,1)=str2num(fulldb(1,pp).speaker(2));
end
elseif combined==1
for pp=1:length(fulldb)
    %conversion of grid left ear -90, in front 0, right 90°
    if fulldb(1,pp).az>180
        fulldb(1,pp).az=fulldb(1,pp).az-360;
    end
    az(pp,1)=fulldb(1,pp).az;
    el(pp,1)=fulldb(1,pp).el;
    speaker(pp,1)=str2num(fulldb(1,pp).speaker(2));
end

else
end

% just median plane
if med==1
    ind_az=find(az==0);
    az=az(ind_az);
    el=el(ind_az);
    fulldb=fulldb(ind_az);

for pp=1:length(fulldb)

    az(pp,1)=fulldb(1,pp).az;
    el(pp,1)=fulldb(1,pp).el;
    speaker(pp,1)=str2num(fulldb(1,pp).speaker(2));
end
end

% 2.)/3.)
az_grid=unique(az);
el_grid=unique(el);
ILD_mat=NaN*ones(length(az_grid), length(el_grid));
ITD_mat=NaN*ones(length(az_grid), length(el_grid));
toa=[];
ILD=[];
```

```

ITD=[];
for nn=1:length(fulldb)

    if ~mod(nn,100)
        disp([num2str(nn/length(fulldb)*100) '%'])
    end

    fulldb(1,nn).ir=resample(fulldb(1,nn).ir(1:N_old,:),16000,fs);
    fulldb(1,nn).ir = fulldb(1,nn).ir.* hrir_window_mat;

    % ILD ITD by HRIR
    % ----- itd estimation -----
    toa(1,:) = onset_detect(fulldb(1,nn).ir(1:100,:), 10, -6);

    fulldb(1,nn).ITD = (toa(1,1)-toa(1,2))/fs*10^6; % just for Interaural channels here..
    ITD(nn,kk)= fulldb(1,nn).ITD;
    ITD_mat(find(az_grid==fulldb(1,nn).az),find(el_grid==fulldb(1,nn).el))=ITD(nn,kk);
    % ----- ild brought band -----
    fulldb(1,nn).ILD = 20*log10(sqrt(mean(fulldb(1,nn).ir(:,1).^2))) - 20*log10(sqrt(mean(
fulldb(1,nn).ir(:,2).^2)));
    ILD(nn,kk)=fulldb(1,nn).ILD;
    ILD_mat(find(az_grid==fulldb(1,nn).az),find(el_grid==fulldb(1,nn).el))=ILD(nn,kk);
end

if plot==1

if ~exist(['verification_plots/' names_HRIR{kk}(1:end-4)] , 'dir')
    mkdir(['verification_plots/' names_HRIR{kk}(1:end-4)])
end

[EL_grid, AZ_grid]=meshgrid(el_grid, az_grid);

% top view ITD
h1=myFigure(18,12,1);
surf(AZ_grid, EL_grid, ITD_mat, 'EdgeColor', 'none');
view([0 90])
colormap(gray(2^9));
cb=colorbar;%('location', 'NorthOutside')
zlab = get(cb,'ylabel');
set(zlab,'String','ITD [\mus]');

xlabel('azimuth \vartheta [\u00b0]');
ylabel('elevation $\varphi$ \lbrack^\circ\rbrack', 'interpreter','latex');
zlabel('ITD [\mus]')
xlim([min(az_grid) max(az_grid)])
ylim([min(el_grid) max(el_grid)])

print(h1,'-dpdf', '-r300', ['verification_plots/' names_HRIR{kk}(1:end-4) '/'

```

```

names_HRIR{kk}(1:end-4) '_ITD_3D_top.pdf'))
saveas(h1, ['verification_plots/' names_HRIR{kk}(1:end-4) '/' names_HRIR{kk}(1:end-4)
'_ITD_3D_top.fig'], 'fig')

view([-45 15])
print(h1, '-dpdf', '-r300', ['verification_plots/' names_HRIR{kk}(1:end-4) '/'
names_HRIR{kk}(1:end-4) '_ITD_3D_side.pdf'])
saveas(h1, ['verification_plots/' names_HRIR{kk}(1:end-4) '/' names_HRIR{kk}(1:end-4)
'_ITD_3D_side.fig'], 'fig')

h2=myFigure(18,12,2);
surf(AZ_grid, EL_grid, ILD_mat, 'EdgeColor', 'none');
view([0 -90])
colormap(gray(2^9));
cb=colorbar;%('location', 'NorthOutside')
zlab = get(cb,'ylabel');
set(zlab,'String','ILD [dB]');

xlabel('azimuth \vartheta [°]');
ylabel('elevation $\varphi$ \lbrack^\circ \rbrack$', 'interpreter','latex');
zlabel('ILD [dB]')
xlim([min(az_grid) max(az_grid)])
ylim([min(el_grid) max(el_grid)])

%
print(h2, '-dpdf', '-r300', ['verification_plots/' names_HRIR{kk}(1:end-4) '/'
names_HRIR{kk}(1:end-4) '_ILD_3D_top.pdf'])
saveas(h2, ['verification_plots/' names_HRIR{kk}(1:end-4) '/' names_HRIR{kk}(1:end-4)
'_ILD_3D_top.fig'], 'fig')

view([-45 15])
print(h2, '-dpdf', '-r300', ['verification_plots/' names_HRIR{kk}(1:end-4) '/'
names_HRIR{kk}(1:end-4) '_ILD_3D_side.pdf'])
saveas(h2, ['verification_plots/' names_HRIR{kk}(1:end-4) '/' names_HRIR{kk}(1:end-4)
'_ILD_3D_side.fig'], 'fig')

%for nn=1:length(fulldb)

%if ~mod(nn,100)
%    disp([num2str(nn/length(fulldb)*100) '%'])
%    end
%
%
%
%
%
%
% h1=figure(2*kk-1);

```

```

%
%
% p1(nn,1)=plot(fulldb(1,nn).az, fulldb(1,nn).ITD, '.', 'MarkerSize',5,'color',
[(fulldb(1,nn).el+abs(min(el)))/(abs(max(el))+abs(min(el))) (fulldb(1,nn).el+abs(min
(el)))/(abs(max(el))+abs(min(el))) (fulldb(1,nn).el+abs(min(el)))/(abs(max(el))+abs(min
(el)))]);
% if nn==1
% xlabel('azimuth $\vartheta$ $\lbrack^\circ\rbrack$')
% ylabel('ITD $\lbrack\mu s\rbrack$')
% set(gcf, 'color',[0.7 0.8 0.8]);
%set(gca, 'color',[0.7 0.8 0.8]);
%
% end
% hold on
%
%
% h3=figure(kk+8);
% p3(nn,1)=plot3(fulldb(1,nn).az, fulldb(1,nn).el, fulldb(1,nn).ITD, '.',
'MarkerSize',5,'color', [(fulldb(1,nn).ITD+abs(min(ITD)))/(abs(max(ITD))+abs(min(ITD)))
(fulldb(1,nn).ITD+abs(min(ITD)))/(abs(max(ITD))+abs(min(ITD))) (fulldb(1,nn).ITD+abs
(min(ITD)))/(abs(max(ITD))+abs(min(ITD)))]);
% if nn==1
% xlabel('azimuth $\vartheta$ $\lbrack^\circ\rbrack$');
% ylabel('elevation $\varphi$ $\lbrack^\circ\rbrack$');
% zlabel('ITD $\lbrack\mu s\rbrack$')
% set(gcf, 'color',[0.7 0.8 0.8]);
%set(gca, 'color',[0.7 0.8 0.8]);
%
% end
% hold on
%
%
% h2=figure(2*kk);
% p2(nn,1)=plot(fulldb(1,nn).az, fulldb(1,nn).ILD, '.',
'MarkerSize',5,'color', [(fulldb(1,nn).el+abs(min(el)))/(abs(max(el))+abs(min(el)))
(fulldb(1,nn).el+abs(min(el)))/(abs(max(el))+abs(min(el))) (fulldb(1,nn).el+abs(min
(el)))/(abs(max(el))+abs(min(el)))]);
%
% if nn==1
% xlabel('azimuth $\vartheta$ $\lbrack^\circ\rbrack$');
% ylabel('ILD $\lbrackdB\rbrack$');
% set(gcf, 'color',[0.7 0.8 0.8]);
%set(gca, 'color',[0.7 0.8 0.8]);
% end
%
% hold on
%
%
%
% h4=figure(kk+12);
% p4(nn,1)=plot3(fulldb(1,nn).az, fulldb(1,nn).el, fulldb(1,nn).ILD, '.',
'MarkerSize',5,'color', [(fulldb(1,nn).ILD+abs(min(ILD)))/(abs(max(ILD))+abs(min(ILD)))
(fulldb(1,nn).ILD+abs(min(ILD)))/(abs(max(ILD))+abs(min(ILD))) (fulldb(1,nn).ILD+abs
(min(ILD)))/(abs(max(ILD))+abs(min(ILD)))]);
% if nn==1

```

```

% xlabel('azimuth $\vartheta$ \lbrack^\circ \rbrack$');
% ylabel('elevation $\varphi$ \lbrack^\circ \rbrack$');
% zlabel('ILD \lbrack$dB$ \rbrack$');
% set(gcf, 'color', [0.7 0.8 0.8]);
%set(gca, 'color', [0.7 0.8 0.8]);
%
% end
% hold on
%
%end

%%
if ~exist(['verification_plots/' names_HRIR{kk}(1:end-4)] , 'dir')
    mkdir(['verification_plots/' names_HRIR{kk}(1:end-4)])
end
%leg_nn=[1:floor(nn/5):nn];
%
%
%
%
h1=figure(2*kk-1);
%legend(p1(leg_nn,1),{['$\varphi$ ' num2str(fulldb(1,leg_nn(1)).el) '$^\circ$'] ✓
['$\varphi$ ' num2str(fulldb(1,leg_nn(2)).el) '$^\circ$'] ['$\varphi$ ' num2str(fulldb(
1,leg_nn(3)).el) '$^\circ$'] ['$\varphi$ ' num2str(fulldb(1,leg_nn(4)).el) '$^\circ$'] ✓
['$\varphi$ ' num2str(fulldb(1,leg_nn(5)).el) ✓
'$^\circ$']},'Interpreter','latex','Location','SouthEast');
%%legend('boxoff')
%
%print(h1,'-dpdf', '-r300', ['verification_plots/' names_HRIR{kk}(1:end-4) '_ITD.pdf'])
%saveas(h1, ['verification_plots/' names_HRIR{kk}(1:end-4) '_ITD.fig'],'fig')
%
%
h2=figure(2*kk);
%legend(p2(leg_nn,1),{['$\varphi$ ' num2str(fulldb(1,leg_nn(1)).el) '$^\circ$'] ✓
['$\varphi$ ' num2str(fulldb(1,leg_nn(2)).el) '$^\circ$'] ['$\varphi$ ' num2str(fulldb(
1,leg_nn(3)).el) '$^\circ$'] ['$\varphi$ ' num2str(fulldb(1,leg_nn(4)).el) '$^\circ$'] ✓
['$\varphi$ ' num2str(fulldb(1,leg_nn(5)).el) ✓
'$^\circ$']},'Interpreter','latex','Location','NorthEast');
%%legend('boxoff')
%
%print(h2,'-dpdf', '-r300', ['verification_plots/' names_HRIR{kk}(1:end-4) '_ILD. ✓
pdf'])
%saveas(h2, ['verification_plots/' names_HRIR{kk}(1:end-4) '_ILD.fig'],'fig')
%
%h3=figure(kk+8);
%print(h3,'-dpdf', '-r300', ['verification_plots/' names_HRIR{kk}(1:end-4) '_ITD_3D. ✓
pdf'])
%saveas(h3, ['verification_plots/' names_HRIR{kk}(1:end-4) '_ITD_3D.fig'],'fig')
%
%h4=figure(kk+12);
%print(h4,'-dpdf', '-r300', ['verification_plots/' names_HRIR{kk}(1:end-4) '_ILD_3D. ✓
pdf'])
%saveas(h4, ['verification_plots/' names_HRIR{kk}(1:end-4) '_ILD_3D.fig'],'fig')
%
%

```



```
end

%%

fs_old=fs;
fs=16000;
if hor==1
save(fullfile(save_path, ['HOR_' names_HRIR{1,kk}(1:end-4) '_' num2str(N) '_mode_' mode✓
'.mat']), 'description', 'channames', 'fs', 'fulldb');
elseif med==1
save(fullfile(save_path, ['MED_' names_HRIR{1,kk}(1:end-4) '_' num2str(N) '_mode_' mode✓
'.mat']), 'description', 'channames', 'fs', 'fulldb');
elseif combined==1
save(fullfile(save_path, ['SPHERE_' names_HRIR{1,kk}(1:end-4) '_' num2str(N) '_mode_' ✓
mode '.mat']), 'description', 'channames', 'fs', 'fulldb');
else
error('error in HRTF_processing: no grid modus chosen, set EITHER hor, med OR ✓
combined to 1');
end
fs=fs_old;

end
```

## ii Ausbreitungsdämpfung

```
function [attenuation_coeff] = ISO96131(fs, nfft, T, pa, h)

% calculate air dumption, pure-tone attenuation coefficient,
% in decibels per metre, for atmospheric absorption
%
% input:
%   fs - sampling frequency
%   nfft - fft length
%   optional:
%   T - temperature [K] default: 293.15 K
%   pa - ambient atmospheric pressure [kPa] default: 101.325 kPa
%   h - humidity [%] default: 50 %
%
% output:
%   attenuation_coeff - attenuation coefficient
%
% Reinhild Roden, Jan. 2015

f=linspace(1/fs,fs/2,nfft/2);

T0 =293.15;           %[K] 20GradCelsius
pr =101.325;          %[kPa]

if ~exist('T', 'var')
    T =293.15;           %[K] 20GradCelsius
end
if ~exist('pa', 'var')
    pa =101.325;          %[kPa]
end
if ~exist('h', 'var')
    h = 50;              % [%] humidity
end

fr0 = (pa/pr)*(24+4.04*10^4*h*((0.02 + h) / (0.391 + h)));
frN = (pa/pr)*(T/T0)^(-1/2)*(9+280*h*exp(-4.170*((T/T0)^(-1/3)-1)));

attenuation_coeff = 8.686.*f.^2.*((1.84.*10.^(-11)).*(pa./pr).^(-1).*(T/T0).^(1/2)) ...
    + (T/T0).^(-5/2) ...
    .*(0.01275.*exp(-2239.1/T).*(fr0 + f.^2/fr0).^(-1) ...
    + 0.1068.*exp(-3352/T).*(frN + f.^2/frN).^(-1))); % [dB/m]
attenuation_coeff=[attenuation_coeff fliplr(attenuation_coeff)];
```

### iii Rauschsignal

```
function noiseOut = GenNoise(noiseLen,type,fs,fl,fh,num)

%% function to generate colored noise
% USAGE:      noiseOut = genNoise(noiseLen,type)
% Input:      noiseLen      length of noise signal in time domain
%             type          'pink','white','brown','blue','violet'
%             fs            sampling frequency
%             fl            lowest frequency
%             fh            highest frequency
%             num           number of channels
%
% Output      sigOut        amplitude modulated signal
%
%
% Author:     Henning Schepker (schehg)   schehg@idmt.fraunhofer.de
%             based on script by Stephan Gerlach (gerlsn)
%             stephan.gerlach@idmt.fraunhofer.de
%
% Update:     2010-08-16      - 1st build (gerlsn)
%             2010-08-25      - add type of noise (pink,white...)
%                               - define lower and upper cut off freq
%                               (schehg)
%             2010-08-31      - number of channels (gerlsn)

n = pow2(nextpow2(noiseLen)); % set to smallest powerof2 > blocksize
noiseOut = zeros(noiseLen,num);

if fh > fs/2,
    fh = fs/2;
end
if fl < 1 , fl = 1; end

for loop=1:1:num
    f = (linspace(0,fs/2,n/2+1)).';
    H = ones(n/2+1,1); % put every frequency into transferfunction
    H(1) = 0; % without DC
    H(n/2+1)=0; % without Nyquistfrequency

    % set transferfunction in unwanted frequencies to 0
    H(1:floor(fl./fs.*n)) = 0;
    H(ceil(fh./fs.*n):end) = 0;

    switch type
        case {'white'}
            H = H; % white noise charakter
        case {'pink'}
            H = H./f;
        case {'brown'}
            H = H./f.^2;
        case {'blue'}
            H = H.*f;
        case {'violet'}
            H = H.*f.^2;
    end
end
```

```
H(isnan(H) | (isinf(H))) = 0; % clean the calculation

% put uniform distributed phase
H = H.*exp(sqrt(-1)*2*pi*rand(n/2+1,1));

% conjugate complex of generated
H = [H(1:((n/2)+1)); conj(flipud(H(2:n/2)))];

% get real part from iFFt
timeH = real(ifft(H,n));
timeH = timeH(1:noiseLen,:);

% put output to 0dB level
timeH = timeH ./ sqrt(mean((timeH.^2)));

noiseOut(:,loop)= timeH;
end
```

**iv   Feature   Extraktion   und   Bildung   des  
Ausgangsdatensatz**

```
% FUNCTION TO GENEARTE INPUT/TARGET-MATFILES FEEDING THE DNN
function [] = prep_dataset_function(binaural, kind_of_signal, radius_vec, speaker_vec, num_catch, kind_of_input, dimension, data_set, mode)
%% setup params
% INPUT
% binaural=1;          - options: 0 = multichannel (no option yet), 1 = binaural
% radius_vec=[1:3];   - numbers from 1 to 16
% speaker_vec=[1:2];  - e.g. [1 2] takes files from speaker 1 and 2
% num_catch=300;      - take num_catch speech files each speaker
%
% setup for DNN-input/chosen features and dimension:
% kind_of_input        - possible strings:
%                        'mfcc'
%                        'mag'
%                        'mag_phase'
%                        'real_imag'
%                        'ILD_ITD_broad'
%                        'ITD_broad'
%                        'ILD_broad'
%                        'ILD_ITD_narrow'
%                        'xcorr_narrow'
%                        'xcorr_broad'
% dimension            - possible strings:
%                        'az'
%                        'el'
%                        'az_el'
%                        'az_dist'
%                        'az_el_dist'
% data_set             - 'BandK_HA'
%                       'HEADacoustics'
%                       'KEMAR_HA'
%                       'SigProc'
% mode                 - 'full'
%                       - 'front'
%                       - 'side'
% Reinhild Roden, reinhild.roden@gmx.net, 02/2015
%% further params

% dataset
switch dimension
case 'az'
    if strcmp(mode, 'front')
        num_hrir_dataset=1;
    elseif strcmp(mode, 'side')
        num_hrir_dataset=3;
    elseif strcmp(mode, 'full')
        num_hrir_dataset=2;
    else
        end
    radius_vec=1;
case 'el'
    if strcmp(mode, 'front')
        num_hrir_dataset=4;
    elseif strcmp(mode, 'side')
        num_hrir_dataset=6;
```



```
elseif strcmp(mode, 'full')
    num_hrir_dataset=5;
else
end
radius_vec=1;
case 'az_el'
    if strcmp(mode, 'front')
        num_hrir_dataset=7;
    elseif strcmp(mode, 'side')
        num_hrir_dataset=9;
    elseif strcmp(mode, 'full')
        num_hrir_dataset=8;
    else
    end
    radius_vec=1;
case 'az_dist'
    if strcmp(mode, 'front')
        num_hrir_dataset=1;
    elseif strcmp(mode, 'side')
        num_hrir_dataset=3;
    elseif strcmp(mode, 'full')
        num_hrir_dataset=2;
    else
    end
case 'az_el_dist'
    if strcmp(mode, 'front')
        num_hrir_dataset=7;
    elseif strcmp(mode, 'side')
        num_hrir_dataset=9;
    elseif strcmp(mode, 'full')
        num_hrir_dataset=8;
    else
    end
otherwise
end
sub_dataset='OLLO20_NO'; %no dialect

% vector length
len_radius_vec=length(radius_vec);
len_speaker_vec=length(speaker_vec);

% grid
radius=[0.25:0.25:4]; % [m]
switch mode
    case 'full'
        res_az=2;
        res_el=2;
        grid_az=-178:res_az:180;
        grid_el=-10:res_el:68;
    case 'front'
        res_az=10;
        res_el=10;
        grid_az=-30:res_az:30;
        grid_el=-10:res_el:50;
    case 'side'
```

```
        res_az=10;
        res_el=10;
        grid_az=60:res_az:120;
        grid_el=-10:res_el:50;
end

% onset params
onset_tresh=-20; % onset defined as first indice under maximum belonging to a value >=✓
onset_tresh
up_samp=10; % upsampling by factor up_samp befor onset calculation, also used for✓
calculating the ITD from xcorr
offset=20; % cut signal at onset_f-20 and onset_b+offset

% window
N_fade=20; %[samples], 1.25 ms
fs=16000; % given by logatom dataset
nfft=512; % fft length

% frame-/hop-size [samples]
if strcmp(kind_of_signal, 'speech')
    frm_ms=25;
    hop_ms=10;
    frm=ceil(0.025*fs); % frame size 25 ms...(a good value for speech 20-30ms)
    hop=ceil(0.01 *fs); % hop size 10 ms
    frm_ILD_ITD=ceil(0.025*fs); %100 ms to get smoother values
    hop_ILD_ITD=ceil(0.01*fs);
    %frm_gamma=ceil(0.025*fs); % frame size 100 ms
    %hop_gamma=ceil(0.01 *fs); % hop size 50 ms
elseif strcmp(kind_of_signal, 'noise')
    % frame-/hop-size [samples]
    frm_ms=1*1000;
    hop_ms=0.4*1000;
    frm=ceil(1*fs); % frame size 25 ms...(a good value for speech 20-30ms)
    hop=ceil(0.4*fs); % hop size 10 ms
    frm_ILD_ITD=ceil(1*fs); %100 ms to get smoother values
    hop_ILD_ITD=ceil(0.4*fs);
    %frm_gamma=ceil(0.025*fs); % frame size 100 ms
    %hop_gamma=ceil(0.01 *fs); % hop size 50 ms
else
    error('set kind_of_signal to speech or noise')
end

% limitation of Gammatone filterbank [Hz]
f_lim=[100 8000];

%% HRIR params, file names
HRIR_path=fullfile(['HRTF_short/' data_set]);
files_HRIR      = dir(HRIR_path);
names_HRIR      = {files_HRIR.name};      names_HRIR(1:2)      =[];
len_hrir_names=length(names_HRIR);

% LOGATOM, file names
folders = dir(fullfile('logatom', sub_dataset)); % read folders including signals✓
spoken without dialect
```

```
names_folders      = {folders.name}; names_folders(1:2)      =[];

%% processing
disp(names_HRIR{1,num_hrir_dataset});
cell_fullldb={};
ir={};
count=0;
OXinput=[];
OXtarget=[];
OXtarget_az=[];
OXtarget_el=[];
OXtarget_az_el=[];

len_rad=length(num2str(max(radius*100)));

%read set of head related impulse responses and required information
load(fullfile(HRIR_path, names_HRIR{1,num_hrir_dataset}), 'fullldb', 'channames', 'description');
cell_fullldb=struct2cell(fullldb);
az(1:length(cell_fullldb(3,1,:)))=cell_fullldb(3,1,:);
el(1:length(cell_fullldb(4,1,:)))=cell_fullldb(4,1,:);

ir(1:length(cell_fullldb(5,1,:)))=cell_fullldb(5,1,:);
len_chan=length(channames);
% if binaural instead of multichannel
if len_chan>2
    if binaural==1
        ir=cell2mat(ir);
        ind=sort([1:len_chan:size(ir,2) 2:len_chan:size(ir,2)]);
        ir=ir(:,ind);
        ir=mat2cell(ir,[size(ir,1)],repmat(2,[1 size(ir,2)/2]));
        channames=channames{1:2};
    end
end

% remove variables not used
clear cell_fullldb
fullldb = rmfield(fullldb,'ir');
fullldb = rmfield(fullldb,'rec_time');
fullldb = rmfield(fullldb,'PosOppSpeaker');
fullldb = rmfield(fullldb,'TempHum');
fullldb = rmfield(fullldb,'speaker');

% 'PRE'-prepare target output of the DNN for azimuth and elevation
target_az =sparse(zeros(length(ir),length(grid_az)));
target_el =sparse(zeros(length(ir),length(grid_el)));
target_az_el =sparse(zeros(length(ir),length(grid_az)*length(grid_el)));
az=[];
el=[];
for jj=1:length(ir)
    az(jj)=fullldb(jj).az;
    el(jj)=fullldb(jj).el;
```

```

% 1 [-30] 2 [-20] 3 [-10] 4 [0] 5 [10] 6 [20] 7 [30]
%bw.
% 1 [60] 2 [70] 3 [80] 4 [90] 5 [100] 6 [110] 7 [120]
target_az(jj, (fulldb(jj).az-grid_az(1))/res_az+1)=1;

% 1 [-10] 2 [0] 3 [10] 4 [20] 5 [30] 6 [40] 7 [50]
target_el(jj, (fulldb(jj).el-grid_el(1))/res_el+1)=1;

% 1 [-30,-10] 2 [-30, 0] 3 [-30,10] 4 [-30,20] 5 [-30,30] 6 [-30,40] 7 ✓
[-30,50]
% 8 [-20,-10] 9 [-20, 0] 10 [-20,10] 11 [-20,20] 12 [-20,30] 13 [-20,40] 14 ✓
[-20,50]
% 15 [-10,-10] 16 [-10, 0] 17 [-10,10] 18 [-10,20] 19 [-10,30] 20 [-10,40] 21 ✓
[-10,50]
% ...
% 43 [30,-10] 44 [30, 0] 45 [30,10] 46 [30,20] 47 [30,30] 48 [30,40] 49 ✓
[30,50]
target_az_el(jj, (((fulldb(jj).az-grid_az(1))/res_az+1)-1)*length(grid_az)+(fulldb(
(jj).el+abs(grid_el(1)))/res_el+1 )=1;
end
target_az=full(target_az);
target_el=full(target_el);
target_az_el=full(target_az_el);

% prepare slope of window for the time signals
sin_slope_mono=(sin(linspace(0,pi/2, N_fade+1)).^2)';
cos_slope_mono=(cos(linspace(0,pi/2, N_fade+1)).^2)';
sin_slope= repmat(sin_slope_mono, [1 size(ir{1},2)]);
cos_slope= repmat(cos_slope_mono, [1 size(ir{1},2)]);

if strcmp(kind_of_signal, 'noise')
    speaker_vec=1;
end

for kk=speaker_vec;%length(names_folders) %loop for all folders of chosen sub dataset ✓
(no dialect), currently 10 for 10 speakers

    switch kind_of_signal
        case 'speech'
            signals = dir(fullfile('logatom', sub_dataset, names_folders{kk})); % read ✓
signals of individual speaker
            names_signals = {signals.name};
            names_signals(1:3) = [];

            % generate random number in order to catch speech files from folders randomly
            %signals_vec=randperm(length(names_signals));
            load('tools/signals_vec.mat');
        case 'noise'
            signals = dir(fullfile('noises')); % read signals of individual speaker
            names_signals = {signals.name};
            names_signals(1:2) = [];

            % generate random number in order to catch speech files from folders randomly

```

```
    signals_vec=1:length(names_signals);

end

for nn=1:num_catch
    if strcmp(kind_of_signal, 'speech')
        included_logatomes{nn}=names_signals{signals_vec(nn)};
        [y fs]=wavread(fullfile('logatom', sub_dataset, names_folders{kk}),
included_logatomes{nn}));
    elseif strcmp(kind_of_signal, 'noise')
        [y fs]=wavread(fullfile('noises', names_signals{signals_vec(nn)}));
    else
        error('set kind_of_signal to speech or noise')
    end

    %find onset in front (f) and at the back (b) of the signal
    onset_f = round(onset_detect(y, up_samp, onset_tresh));
    onset_b = round(onset_detect(flipud(y), up_samp, onset_tresh));
    %delete useless samples

    if onset_f>offset
        y(1:onset_f-offset)=[];
    end

    if onset_b>offset
        y(end-onset_b+offset:end)=[];
    end

    % use window
    y_slope=ones(length(y),1);
    y_slope(1:N_fade+1) =sin_slope_mono;
    y_slope(end-N_fade:end) =cos_slope_mono;
    y=y.*y_slope;

    % y already mono signal
    for rr=radius_vec
        count=count+1; % for displaying the progress

    % ----- start of preprocessing -----
        if strcmp(dimension, 'az_dist') || strcmp(dimension, 'az_el_dist')
            % 'PRE'-prepare target output of the DNN for distance
            r_fac= (radius(rr)/0.25-1)/4; % 0.25 m distance while recording
            dist = radius(rr)*100; % curr. radius [cm]
            target_dist=sparse(zeros(1,length(radius))); target_dist(rr)=1;

            %-----air dumping-----
            len_y=length(y);
            pow=nextpow2(len_y); % here required fft-length
            [attenuation_coeff] = ISO96131(fs,2^pow); % calc attenuation

            Y=fft(y,2^pow); % calc fft of speech signal
            Y_abs=abs(Y);
            Y_wrap_angle=angle(Y);
```

```

        Y_abs=Y_abs.*10.^(-attenuation_coeff'.*r_fac/20); % apply attenuation
        Y=Y_abs.*exp(i*Y_wrap_angle); % recompose spectrum
        y=real(ifft(Y,2^pow)); % back to time signal
        y=y(1:len_y); % avoid zeros
    end

    % convolve with all HRIRs
    y_conv =cellfun(@conv2, repmat({y}, [1 length(ir)]), ir, 'UniformOutput', false); %✓
    conv2(y, ir{jj});

    % window
    hrir_window = ones(size(y_conv{1}));
    hrir_window(1:N_fade+1,:) = sin_slope;
    hrir_window(end-N_fade:end,:) = cos_slope;
    y_conv=cellfun(@(v) v.*hrir_window, y_conv, 'UniformOutput', false);

    % ----- get input/target for the DNN -----

    % get short term fourier transform if necessary
    if strcmp(kind_of_input, 'mag') || strcmp(kind_of_input, 'mag_phase') || strcmp(
(kind_of_input, 'real_imag')
        [SPEC,nfft_curr]=get_stft(y_conv,fs,frm,hop); % cell: 1 x nFrames: 257 x✓
        gridp_points*2

        fac=nfft_curr/nfft;
        fac_round=round(fac);
        if fac>=1 && ((fac-fac_round)==0)

            SPEC=cellfun(@(v) v(1:fac:end,:), SPEC, 'UniformOutput', false);
        else
            error('error: false nfft')
        end

        nFrames=size(SPEC,2);
        input_length=size(SPEC{1},1)*2;
        grid_points=size(SPEC{1},2)/2;

        % reshape cells:
        % 1.) cell: 1 x nFrames: 514 x grid_points
        SPEC=cellfun(@reshape, SPEC, repmat({input_length}, [1 nFrames]), repmat(
({grid_points}, [1 nFrames]), 'UniformOutput', false);
        % 2.) cell: nFrames x 1: grid_points x 514
        SPEC=(cellfun(@(v) v', SPEC, 'UniformOutput', false))';
        % 3.) double: grid_points*nFrames x 514
        SPEC=cell2mat(SPEC);
    end

    % build input matrix
    switch kind_of_input
    case 'mfcc'
        options=struct; % take default parameters
        y_conv_mat=cell2mat(y_conv);
        y_conv_cell_left=mat2cell(y_conv_mat(:,1:2:end), [size(y_conv_mat, 1)],✓

```

```

repmat(1, [1 size(y_conv_mat, 2)/2]));
    y_conv_cell_right=mat2cell(y_conv_mat(:,2:2:end), [size(y_conv_mat, 1)],
repmat(1, [1 size(y_conv_mat, 2)/2]));

    %c=melcepst(s,fs,w,nc,p,n,inc,fl,fh)
    [mfcc_left_voicebox] = cellfun(@melcepst, y_conv_cell_left, repmat({fs},
[1 size(y_conv_cell_left, 2)]), repmat({'M'}, [1 size(y_conv_cell_left, 2)]), repmat
({12}, [1 size(y_conv_cell_left, 2)]), repmat({floor(3*log(fs))}, [1 size
(y_conv_cell_left, 2)]), repmat({frm}, [1 size(y_conv_cell_left, 2)]), repmat({hop}, [1
size(y_conv_cell_left, 2)]), 'UniformOutput', false);
    [mfcc_right_voicebox] = cellfun(@melcepst, y_conv_cell_right, repmat({fs},
[1 size(y_conv_cell_right, 2)]), repmat({'M'}, [1 size(y_conv_cell_right, 2)]), repmat
({12}, [1 size(y_conv_cell_right, 2)]), repmat({floor(3*log(fs))}, [1 size
(y_conv_cell_right, 2)]), repmat({frm}, [1 size(y_conv_cell_right, 2)]), repmat({hop}, [1
size(y_conv_cell_right, 2)]), 'UniformOutput', false);
    mfcc_left=cellfun(@(v) v', mfcc_left_voicebox, 'UniformOutput', false);
    mfcc_right=cellfun(@(v) v', mfcc_right_voicebox, 'UniformOutput', false);

    nFrames=size(mfcc_left{1,1},2);

    % reshape mat:
    mfcc_left=(reshape(cell2mat(mfcc_left'),length(mfcc_left{1,1}(:,1)),
length(mfcc_left{1,1}(1,:))*length(mfcc_left))');
    mfcc_right=(reshape(cell2mat(mfcc_right'),length(mfcc_right{1,1}(:,1)),
length(mfcc_right{1,1}(1,:))*length(mfcc_right))');

    OXinput=[OXinput; mfcc_left mfcc_right];

case 'mag'
    OXinput=[OXinput; abs(SPEC)];
case 'mag_phase'
    OXinput=[OXinput; abs(SPEC) angle(SPEC)];
case 'real_imag'
    OXinput=[OXinput; real(SPEC) imag(SPEC)];
case 'ILD_ITD_broad'
    % ILD/ITD_broad: cell: 1 x nFrames: 1 x grid_points
    [ILD_broad, ITD_broad, ~,~,~,~]=get_gamma_feature(y_conv,fs,[1 1 0 0 0
0], frm_ILD_ITD, hop_ILD_ITD, f_lim);

    nFrames=size(ILD_broad,2);
    grid_points=size(ILD_broad{1},2);
    % reshape cells:
    % 1.) cell: 1 x nFrames: 1 x grid_points
    ILD_broad=cellfun(@reshape, ILD_broad, repmat({grid_points}, [1
nFrames]), repmat({1}, [1 nFrames]), 'UniformOutput', false);
    ITD_broad=cellfun(@reshape, ITD_broad, repmat({grid_points}, [1
nFrames]), repmat({1}, [1 nFrames]), 'UniformOutput', false);

    % double: nFrames*grid_points x 1
    ILD_broad=cell2mat(ILD_broad');
    ITD_broad=cell2mat(ITD_broad');

    OXinput=[OXinput; ILD_broad ITD_broad];

```

```
case 'ILD_broad'
    % ILD/ITD_broad: cell: 1 x nFrames: 1 x grid_points
    [ILD_broad,~, ~,~,~,~]=get_gamma_feature(y_conv,fs,[1 0 0 0 0 0]),✓
frm_ILD_ITD, hop_ILD_ITD, f_lim);

    nFrames=size(ILD_broad,2);
    grid_points=size(ILD_broad{1},2);
    % reshape cells:
    % 1.) cell: 1 x nFrames: 1 x grid_points
    ILD_broad=cellfun(@reshape, ILD_broad, repmat({grid_points}, [1✓
nFrames]), repmat({1}, [1 nFrames]), 'UniformOutput', false);

    % double: nFrames*grid_points x 1
    ILD_broad=cell2mat(ILD_broad');

    OXinput=[OXinput; ILD_broad];
case 'ITD_broad'
    % ILD/ITD_broad: cell: 1 x nFrames: 1 x grid_points
    [~, ITD_broad, ~,~,~,~]=get_gamma_feature(y_conv,fs,[0 1 0 0 0 0]),✓
frm_ILD_ITD, hop_ILD_ITD, f_lim);

    nFrames=size(ITD_broad,2);
    grid_points=size(ITD_broad{1},2);
    % reshape cells:
    % 1.) cell: 1 x nFrames: 1 x grid_points
    ITD_broad=cellfun(@reshape, ITD_broad, repmat({grid_points}, [1✓
nFrames]), repmat({1}, [1 nFrames]), 'UniformOutput', false);

    % double: nFrames*grid_points x 1
    ITD_broad=cell2mat(ITD_broad');

    OXinput=[OXinput; ITD_broad];

case 'ILD_ITD_narrow'
    % cell: nFrames x 1: double: grid_points x ERB
    [~,~, ILD_narrow, ITD_narrow,~,~]=get_gamma_feature(y_conv,fs,[0 0 1 1 0 0✓
0], frm_ILD_ITD, hop_ILD_ITD, f_lim);

    nFrames=size(ILD_narrow,1);
    grid_points=size(ILD_narrow{1},1);
    % double: nFrames*grid_points x ERB
    ILD_narrow=cell2mat(ILD_narrow);
    ITD_narrow=cell2mat(ITD_narrow);

    OXinput=[OXinput; ILD_narrow ITD_narrow];
case 'ITD_narrow'
    % cell: nFrames x 1: double: grid_points x ERB
    [~,~,~, ITD_narrow,~,~]=get_gamma_feature(y_conv,fs,[0 0 0 1 0 0]),✓
frm_ILD_ITD, hop_ILD_ITD, f_lim);

    nFrames=size(ITD_narrow,1);
    grid_points=size(ITD_narrow{1},1);
    % double: nFrames*grid_points x ERB
```



```

        ITD_narrow=cell2mat(ITD_narrow);

        OXinput=[OXinput;ITD_narrow];
    case 'ILD_narrow'
        % cell: nFrames x 1: double: grid_points x ERB
        [~,~, ILD_narrow,~,~,~]=get_gamma_feature(y_conv,fs,[0 0 1 0 0 0]),
        frm_ILD_ITD, hop_ILD_ITD, f_lim);

        nFrames=size(ILD_narrow,1);
        grid_points=size(ILD_narrow{1},1);
        % double: nFrames*grid_points x ERB
        ILD_narrow=cell2mat(ILD_narrow);

        OXinput=[OXinput; ILD_narrow] ;

    case 'xcorr_narrow'
        % cell: nFrames x 1: double: grid_points x 637*ERB
        [~,~,~,~, GAMMA_xcorr,~]=get_gamma_feature(y_conv,fs,[0 0 0 0 1 0]), frm,
        hop, f_lim);

        nFrames=size(GAMMA_xcorr,1);
        grid_points=size(GAMMA_xcorr{1},1);
        % double: nFrames*grid_points x 637*ERB
        GAMMA_xcorr=cell2mat(GAMMA_xcorr);

        OXinput=[OXinput; GAMMA_xcorr];
    case 'xcorr_broad'
        % cell: nFrames x 1: double: grid_points x 637*ERB
        [~,~,~,~,broad_xcorr]=get_gamma_feature(y_conv,fs,[0 0 0 0 0 1]), frm,
        hop, f_lim);

        nFrames=size(broad_xcorr,2);
        grid_points=size(broad_xcorr{1},2);
        % double: nFrames*grid_points x 637
        broad_xcorr=(cell2mat(broad_xcorr));

        OXinput=[OXinput; broad_xcorr];
    otherwise
        error('kind of input not known')
end

% build traget matrix
switch dimension
    case 'az'
        tmp= repmat(target_az, [nFrames 1]);
        OXtarget=[OXtarget; tmp];
    case 'el'
        tmp= repmat(target_el, [nFrames 1]);
        OXtarget=[OXtarget; tmp];
    case 'az_el'
        tmp_az= repmat(target_az, [nFrames 1]);
        tmp_el= repmat(target_el, [nFrames 1]);
        tmp_az_el= repmat(target_az_el, [nFrames 1]);
        OXtarget=[OXtarget; tmp_az tmp_el];

```

```

        OXtarget_az=[OXtarget_az; tmp_az];
        OXtarget_el=[OXtarget_el; tmp_el];
        OXtarget_az_el=[OXtarget_az_el; tmp_az_el];
    case 'az_dist'
        tmp_az= repmat(target_az, [nFrames 1]);
        tmp_dist= repmat(target_dist, [grid_points*nFrames 1]);

        OXtarget=[OXtarget; tmp_az tmp_dist];
    case 'az_el_dist'
        tmp_az= repmat(target_az, [nFrames 1]);
        tmp_el= repmat(target_el, [nFrames 1]);
        tmp_dist= repmat(target_dist, [grid_points*nFrames 1]);

        OXtarget=[OXtarget; tmp_az tmp_el tmp_dist];
    otherwise
        error('pair of dimesnions not known');
    end

    % display progress
    disp([kind_of_input ' ' num2str(count) ' of ' num2str(
(len_speaker_vec*num_catch*len_radius_vec) ' : ' num2str(
(count/len_speaker_vec/num_catch/len_radius_vec*100) '% done' ]])
    end
end
end
%%% eval
%az=az';
%nfr=size(OXinput,1)/grid_points;
%az_long= repmat(az, [nfr 1]);
%[~, ind]=sort(az_long);
%figure(1)
%plot(az_long(ind), OXinput(ind,1), '*')
%figure(2)
%plot(az_long(ind), OXinput(ind,2), '*')

%% build testing and training dataset,

% split data in testing and training datasets
OXtestInput=OXinput(1:round(size(OXinput,1)/2),:); OXinput(1:round(size(OXinput,1)
/2),:)=[];
OXtestTarget=OXtarget(1:round(size(OXtarget,1)/2),:); OXtarget(1:round(size(OXtarget,1)
/2),:)=[];

if strcmp(dimension,'az_el')
    OXtestTarget_az=OXtarget_az(1:round(size(OXtarget_az,1)/2),:); OXtarget_az(1:round
(size(OXtarget_az,1)/2),:)=[];
    OXtestTarget_el=OXtarget_el(1:round(size(OXtarget_el,1)/2),:); OXtarget_el(1:round
(size(OXtarget_el,1)/2),:)=[];
    OXtestTarget_az_el=OXtarget_az_el(1:round(size(OXtarget_az_el,1)/2),:);
OXtarget_az_el(1:round(size(OXtarget_az_el,1)/2),:)=[];
end

% shuffle data
shuffle_vec_train=randperm(size(OXinput,1));

```

```
shuffle_vec_test=randperm(size(OXtestInput,1));

OXinput=OXinput(shuffle_vec_train,:);
OXtarget=OXtarget(shuffle_vec_train,:);
if strcmp(dimension,'az_el')
    OXtarget_az=OXtarget_az(shuffle_vec_train,:);
    OXtarget_el=OXtarget_el(shuffle_vec_train,:);
    OXtarget_az_el=OXtarget_az_el(shuffle_vec_train,:);
end

OXtestInput=OXtestInput(shuffle_vec_test,:);
OXtestTarget=OXtestTarget(shuffle_vec_test,:);
if strcmp(dimension,'az_el')
    OXtestTarget_az=OXtestTarget_az(shuffle_vec_test,:);
    OXtestTarget_el=OXtestTarget_el(shuffle_vec_test,:);
    OXtestTarget_az_el=OXtestTarget_az_el(shuffle_vec_test,:);
end

%% verifivation plots
% make folder for saving plots

if ~exist('verification_plots', 'dir')
    mkdir('verification_plots')
end

if strcmp(dimension, 'az_el')
    [~,ix]=max(OXtarget_az,[],2);
    h1=myFigure(18,12);
    plot(ix,OXinput,'*');
else
    [~,ix]=max(OXtarget,[],2);
    h1=myFigure(18,12);
    plot(ix,OXinput,'k*');
    hold on
    az=unique(ix);
    OXinput_mean=[];
    for kk=1:length(az)
        OXinput_mean(kk,:)=prctile(OXinput(find(ix==az(kk))),[5 25 50 75 95]);
    end
    plot(grid_az,OXinput_mean(:,3),'g'); hold on
    plot(grid_az,OXinput_mean(:, [1 5]),'r');
    plot(grid_az,OXinput_mean(:, [2 4]),'b');
    legend({kind_of_input, 'mean', '5%, 95% percentile', '25%, 75% percentile'});
    xlabel('azimuth [Å]')
    ylabel('ILD (broad band) [dB]')
    set(h1, 'xticks', grid_az);
    set(h1, 'xticklabel', grid_az)

    print(h1,'-dpdf', '-r300', [dimension '_' kind_of_input{1}])
    saveas(h1, [dimension '_' kind_of_input{1}], 'fig')
end

%% save relevant information
% make folder for saving files
if strcmp(kind_of_signal, 'speech')
```

```

    if binaural==1
        save_path=fullfile('dnn_input_output_speech_test', data_set , 'binaural', mode);
    elseif binaural==0
        save_path=fullfile('dnn_input_output_speech_test', data_set , 'multichan', mode);
    else
        error('error in prep_dtaset: value "binaural" should be set to be 1 or 0')
    end
    if ~exist(save_path, 'dir')
        mkdir(save_path)
    end

    % save files
    if strcmp(dimension, 'az_el')
        save(fullfile(save_path, [dimension '_' kind_of_input '_speaker' num2str(len_speaker_vec)
        '_filenum' num2str(num_catch)]), 'OXinput', 'OXtarget', 'OXtarget_az', 'OXtarget_el', 'OXtarget_az_el', 'OXtestInput',
        'OXtestTarget', 'OXtestTarget_az', 'OXtestTarget_el', 'OXtestTarget_az_el', 'included_logatomes', 'radius_vec', 'speaker_vec', '-v7.3');
    else
        save(fullfile(save_path, [dimension '_' kind_of_input '_speaker' num2str(len_speaker_vec)
        '_filenum' num2str(num_catch)]), 'OXinput', 'OXtarget', 'OXtestInput', 'OXtestTarget', 'included_logatomes', 'radius_vec', 'speaker_vec', '-v7.3');
    end

elseif strcmp(kind_of_signal, 'noise')

    if binaural==1
        save_path=fullfile('dnn_input_output_noise_test', data_set , 'binaural', mode);
    elseif binaural==0
        save_path=fullfile('dnn_input_output_noise_test', data_set , 'multichan', mode);
    else
        error('error in prep_dataset: value "binaural" should be set to be 1 or 0')
    end
    if ~exist(save_path, 'dir')
        mkdir(save_path)
    end

    % save files
    if strcmp(dimension, 'az_el')
        save(fullfile(save_path, [dimension '_' kind_of_input '_noise']), 'OXinput', 'OXtarget', 'OXtarget_az', 'OXtarget_el', 'OXtarget_az_el', 'OXtestInput',
        'OXtestTarget', 'OXtestTarget_az', 'OXtestTarget_el', 'OXtestTarget_az_el', 'radius_vec', '-v7.3');
    else
        save(fullfile(save_path, [dimension '_' kind_of_input '_noise']), 'OXinput', 'OXtarget', 'OXtestInput', 'OXtestTarget', 'radius_vec', '-v7.3');
    end
else
    error('set kind_of_signal to speech or noise')
end

```

```
function [SPEC,nfft] = get_stft( x,fs, frameInSamp, hopInSamp, nfft)
% mean blockwise fft
%
% input:
% x - time signal
% fs - sampling rate
% hopInSamp - hop size in samples
% frameInSamp - frame size in samples
%
% output:
% spec - complex valued spectrum

%% Check for errors...
if ~exist('x', 'var') || ~exist('fs', 'var')
    error('Error in get_mean_blockwise_fft: input vector x and sampling rate fs are
required...');
end

%% default values

if ~exist('frameInSamp', 'var')

    frameInSamp = round(0.025*fs);
end

if ~exist('hopInSamp', 'var')
    hopInSamp = round(0.01*fs);
end

if ~exist('winoption', 'var')
    winoption = 'hanning';
end

nfft = 2^nextpow2(frameInSamp);

%% choose window
x_mat=cell2mat(x);
switch winoption
    case 'hanning'
        WIN = repmat(hanning(frameInSamp), [1 size(x_mat,2)]);
    case 'hamming'
        WIN = repmat(hamming(frameInSamp), [1 size(x_mat,2)]);
    case 'rect'
        WIN = repmat(ones(frameInSamp,1), [1 size(x_mat,2)]);
    otherwise
        error('Unknown window')
end

%% Prepare blockwise calculation...
nFrames = 1 + floor((size(x{1},1)-frameInSamp)/hopInSamp );
% preallocate output vector
```

```
%SPEC = cell(size(x)); SPEC=cellfun(@(v) zeros(nfft/2+1, size(x{1},2), nFrames), SPEC, ✓
'UniformOutput', false);
%SPEC_mat=zeros(nfft/2+1, size(x_mat,2), nFrames);
%% Iterate over all frames and calculate spectrum

%
% decomposition of input signal into parts of size blocklength
decompM = zeros(frameInSamp, nFrames);
currFrame={};
for ii=1:nFrames
    decompM(:, ii) = (1:frameInSamp) + (ii-1)*hopInSamp;
    % get the data of the current frame.*window
    currFrame{ii} = x_mat(decompM(:,ii),:).*WIN;
end

% compute magnitude/phase of block
Y=cellfun(@fft, currFrame, repmat({nfft}, [1 nFrames]), 'UniformOutput', false);
SPEC=cellfun(@(v) v(1:nfft/2+1,:), Y, 'UniformOutput', false);

%SPEC=mat2cell(SPEC_mat, [nfft/2+1], [repmat(size(x{1},2), [1 size(x_mat,2)/size(x{1}, ✓
2)]]], [nFrames]);

end
```

```

function [ILD_broad, ITD_broad, ILD_narrow, ITD_narrow, GAMMA_xcorr, broad_xcorr] = ✓
get_gamma_feature( x,fs, calc_option, frameInSamp, hopInSamp, f_lim, up_samp)
% % time signal x is filtered with an ERB filterbank, the number of bands being
% calculated with the following equation:
%  $n = 21.4 * \log_{10}(0.004367 * f_{up} + 1) - 21.4 * \log_{10}(0.004367 * f_{low} + 1)$  ( $f_{up}$  =
% upper edge frequency,  $f_{low}$  = lower edge frequency)
%
% input:
% x - time signal
% fs - sampling rate
% calc_option - 1st entry: broad band ILD per frame [dB]
%                2nd      : broad band ITD per frame [ys]
%                3rd      : narrow band ILD per frame [dB]
%                4th      : narrow band ITD per frame [ys]
%                5th      : narrow binaural xcorr (multichannel has to be added)
%                6th      : broad binaural xcorr
%
% hopInSamp - hop size in samples
% frameInSamp - frame size in samples
% f_lim      - upper and lower frequency bounds in Hz.
%                (Default = [50 20000])
% up_samp    - factor for upsampling
%
% OUTPUT
% ILD_broad  - broad band ILD per frame [dB]
% ITD_broad  - broad band ITD per frame [ys]
% ILD_narrow - narrow band ILD per frame [dB]
% ITD_narrow - narrow band ITD per frame [ys]
% GAMMA_xcorr - cross correlation of ERB filtered time signals
% broad_xcorr - cross correlation of time signals
%
% see also: MakeERBFilters, ERBFilterBank, erb_error
%
% Reinhild Roden, reinhild.roden@gmx.net, 02/2015

%% Check for errors...
if ~exist('x', 'var') || ~exist('fs', 'var')
    error('Error in get_gamma_time_frames_cellfun: input vector x and sampling rate fs ✓
are required...');
end

%% default values

if ~exist('frameInSamp', 'var')

    frameInSamp = round(0.01*fs);
end

if ~exist('hopInSamp', 'var')
    hopInSamp = round(0.005*fs);
end

if ~exist('f_lim', 'var')
    f_lim = [50 20000];
end

```

```
if ~exist('up_samp', 'var')
    up_samp = 10;
end

%if ~exist('winoption', 'var')
%winoption = 'hanning';
%end

%%
x_mat=cell2mat(x);
size_x_mat = size(x_mat,2); % ir num. of channels

%% choose window

%switch winoption
%    case 'hanning'
%        WIN = repmat(hanning(frameInSamp), [1 size(x_mat,2)]);
%    case 'hamming'
%        WIN = repmat(hamming(frameInSamp), [1 size(x_mat,2)]);
%    case 'rect'
%        WIN = repmat(ones(frameInSamp,1), [1 size(x_mat,2)]);
%    otherwise
%        error('Unknown window')
%end

%% 1. prepare filtering
if ~(calc_option(2)==1 || calc_option(1)==1 || calc_option(6)==1)
% get ERB filter coefficients
n = round(21.4*log10(0.004367*f_lim(2)+1)-21.4*log10(0.004367*f_lim(1)+1));
[fcoefs,f_c] = MakeERBFilters(fs,n,f_lim(1));

% sort with ascending frequency
fcoefs = flipud(fcoefs);
f_c     = flipud(f_c);

% delete filters above upper f_lim
n       = find(f_c <= f_lim(2), 1, 'last');
f_c     = f_c(1:n);
fcoefs = fcoefs(1:n,:);

% get filter impulse responses
imp = zeros(frameInSamp,1);
imp(1) = 1;
gamma_ir = ERBFilterBank(imp,fcoefs);
gamma_ir=gamma_ir';
gamma_ir=mat2cell(gamma_ir, [frameInSamp], repmat(1, [1 n]));
end

%% Prepare blockwise calculation...
nFrames = 1 + floor((size(x{1},1)-frameInSamp)/hopInSamp); %niko
if nFrames<=0
    nFrames=1;
    x_mat=[x_mat; zeros(frameInSamp-size(x_mat,1),size(x_mat,2))];
```



```

end
% nFrames = ceil((size(x_mat,1)-frameInSamp+hopInSamp) / hopInSamp);

% preallocate output vector
conv_length=frameInSamp*2-1;
%GAMMA = cell(size(x)); GAMMA=cellfun(@(v) zeros(2*(conv_length)-1, n, nFrames), GAMMA,
'UniformOutput', false);
%GAMMA_mat=zeros(2*(conv_length)-1,size_x_mat/2*n, nFrames);

%% Iterate over all frames and get gammatone filtered timesignal
%
% decomposition of input signal into parts of size blocklength
decompM = zeros(frameInSamp, nFrames);
for ii=1:nFrames
    decompM(:, ii) = (1:frameInSamp) + (ii-1)*hopInSamp;
end
count=0;

for ii = 1:nFrames
    % get the data of the current frame
    %if sqrt(mean(x_mat(decompM(:,ii),:).^2))>=10^(-0/20) & sqrt(mean(x_mat(decompM(:,ii),:).^2))<=0.9;% 0.01*max(max(x_mat))%0.0001
    count=count+1;
    currFrame{count} = x_mat(decompM(:,ii),:);%.*WIN;
    frame_left{count}=x_mat(decompM(:,ii),1:2:end);
    frame_right{count}=x_mat(decompM(:,ii),2:2:end);
    %end
end
nFrames=count;
chan_num=size(x{1},2);
gridpoints=size(x,2);

%%
if calc_option(2)==1 || calc_option(1)==1 || calc_option(6)==1
    %broad band ITD
    frame_left_br=mat2cell(cell2mat(frame_left), [frameInSamp], ones(1,
gridpoints*count));
    frame_right_br=mat2cell(cell2mat(frame_right), [frameInSamp], ones(1,
gridpoints*count));

    %frame_left_br_up=cellfun(@interp, frame_left_br, repmat({up_samp}, [1
gridpoints*count]), 'UniformOutput', false);
    %frame_right_br_up=cellfun(@interp, frame_right_br, repmat({up_samp}, [1
gridpoints*count]), 'UniformOutput', false);
    if calc_option(2)==1 || calc_option(6)==1
        xcorr_broad=mat2cell(cell2mat(cellfun(@xcorr, frame_left_br,frame_right_br,
'UniformOutput', false)),length(frame_right_br{1})*2-1, repmat(gridpoints, [1 count]));
        %xcorr_broad=cellfun(@interp, xcorr_broad, repmat({up_samp}, [1 count]),
'UniformOutput', false);

        [~,ind]=cellfun(@max, xcorr_broad,'UniformOutput', false);
        ind = cellfun(@(v) (v-length(frame_left_br{1})), ind,'UniformOutput', false);
    end
end

```

```
% check for injured ITDs ...if it is longer than a normal ITD

% if there is a head with an interaural distance of 30 cm (conservative
% estimation) and speed of sound is 330 m/s (conservative estimation),
% then ITD can't be bigger than 0.3/330 seconds here!
% is there any frame including a maximum ITD bigger than that it should
% be deleted

ind_max=cell2mat(cellfun(@(v) max(abs(v)), ind, 'UniformOutput', false));
ind_del=find(ind_max>(0.3/330*fs*up_samp));

ind(ind_del)=[];

if calc_option(2)==1
    %ITD_broad = cellfun(@(v) v./(up_samp*fs).*10^6, ind,'UniformOutput', false); % for✓
up sampled data
    ITD_broad = cellfun(@(v) v./fs.*10^6, ind,'UniformOutput', false);

else
    ITD_broad={};
end
else
    ITD_broad={};
end

if calc_option(6)==1
    broad_xcorr =xcorr_broad;
else
    broad_xcorr={};
end
if calc_option(1)==1

    %delete ITD
    %ind_f_l=round(cell2mat(ind));ind_f_l(ind_f_l<0)=0; ind_f_l=ind_f_l(:)';✓
ind_f_l=mat2cell(ind_f_l, 1, ones(1,length(ind_f_l)));
    %ind_b_l=round(cell2mat(ind));ind_b_l(ind_b_l>0)=0; ind_b_l=abs(ind_b_l(:)');✓
ind_b_l=mat2cell(ind_b_l, 1, ones(1,length(ind_b_l)));

    %ind_f_r=ind_b_l;
    %ind_b_r=ind_f_l;

    %frame_left_del=cellfun(@(v,u,w) v(u+1:end-w), frame_left_br, ind_f_l, ind_b_l,✓
'UniformOutput', false);
    %frame_right_del=cellfun(@(v,u,w) v(u+1:end-w), frame_right_br, ind_f_r, ind_b_r,✓
'UniformOutput', false);

%ILD works in a frequency range above 1kHz, highpass indicated

% a Chebyshev Type I IIR filter with a passband
% ripple of 0.5 dB and a 3 dB cutoff frequency at 1000 Hz.

%d = fdesign.highpass('N,F3dB,Ap', 10, 1000, .5, fs);
%d=fdesign.highpass('N,Fc',12,1000,fs);
%Hd = design(d);
```

```

    %frame_left_del_filt = cellfun(@filter, repmat({Hd}, [1 length(frame_left_del)]), ✓
frame_left_del, 'UniformOutput', false);
    %frame_right_del_filt = cellfun(@filter, repmat({Hd}, [1 length(frame_right_del)]), ✓
frame_right_del, 'UniformOutput', false);

    % broad band ILD
    frame_left_sqr=cellfun(@(v) v.^2, frame_left_br, 'UniformOutput', false);
    frame_right_sqr=cellfun(@(v) v.^2, frame_right_br, 'UniformOutput', false);

    frame_left_sqr_mean=cellfun(@mean, frame_left_sqr, repmat({1}, [1 ✓
gridpoints*count]), 'UniformOutput', false);
    frame_right_sqr_mean=cellfun(@mean, frame_right_sqr, repmat({1}, [1 ✓
gridpoints*count]), 'UniformOutput', false);

    frame_left_dB=cellfun(@(v) 20*log10(v.^(1/2)), frame_left_sqr_mean, 'UniformOutput', ✓
false);
    frame_right_dB=cellfun(@(v) 20*log10(v.^(1/2)), ✓
frame_right_sqr_mean, 'UniformOutput', false);

    ILD_broad=cellfun(@(v,u) v-u, frame_left_dB, frame_right_dB, 'UniformOutput', ✓
false);
    ILD_broad=mat2cell(cell2mat(ILD_broad), 1, repmat(gridpoints, [1 count]));
    clear ind
else
    ILD_broad={};
end
%%
if calc_option(3)==1 || calc_option(4)==1 || calc_option(5)==1
    frame_left=cell2mat(frame_left);
    frame_left=repmat({frame_left}, [1 n]);

    frame_right=cell2mat(frame_right);
    frame_right=repmat({frame_right}, [1 n]);

    % filter frame with filerbank
    GAMMA_left=mat2cell(cell2mat(cellfun(@conv2, gamma_ir, frame_left, 'UniformOutput', ✓
false)), [conv_length], ones(1, n*gridpoints*count));
    GAMMA_right=mat2cell(cell2mat(cellfun(@conv2, gamma_ir, frame_right, ✓
'UniformOutput', false)), [conv_length], ones(1, n*gridpoints*count));
end

if calc_option(3)==1
    % narrow band ILD
    ILD_narrow=20*log10(sqrt(mean(cell2mat(GAMMA_left).^2))) - 20*log10(sqrt(mean ✓
(cell2mat(GAMMA_right).^2)));
    ILD_narrow=reshape(ILD_narrow, count*gridpoints, n);
    ILD_narrow=mat2cell(ILD_narrow, repmat(gridpoints, [1 count]), n);
else
    ILD_narrow={};
end
%%
if calc_option(4)==1 || calc_option(5)==1
    GAMMA_xcorr_cell=cellfun(@xcorr, GAMMA_left, GAMMA_right, 'UniformOutput', false);

```

```
end
if calc_option(4)==1
    % narrow ITD via xcorr
    %GAMMA_xcorr_up=cell2mat(cellfun(@interp,GAMMA_xcorr_cell, repmat({up_samp}, [1 length(GAMMA_xcorr_cell)]), 'UniformOutput', false));
    %[~, ind]=max(GAMMA_xcorr_up);
    [~, ind]=max(cell2mat(GAMMA_xcorr_cell));
    %ITD_narrow=ind./up_samp./fs.*10.^(6);
    ITD_narrow=(ind-repmat(length(GAMMA_left{1}), [1 length(ind)]).)/fs.*10.^(6);
    ITD_narrow=reshape(ITD_narrow,count*gridpoints, n);
    ITD_narrow=mat2cell(ITD_narrow, repmat(gridpoints, [1 count]), n);
else
    ITD_narrow={};
end
% !!!!!!!! attention: just binaural output for xcorr yet !!!!!
% !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
%%
if calc_option(5)==1
    % downsampling by factor 10....otherwise inputvecor becomes too big
    GAMMA_xcorr=cell2mat(cellfun(@downsample, GAMMA_xcorr_cell, repmat({10}, [1 length(GAMMA_xcorr_cell)]), 'UniformOutput', false));
    %GAMMA_xcorr=cell2mat(GAMMA_xcorr_cell);
    GAMMA_xcorr_len=size(GAMMA_xcorr,1);
    GAMMA_xcorr=cell2mat((mat2cell(GAMMA_xcorr', repmat(gridpoints*count, [1 n]), GAMMA_xcorr_len))');
    GAMMA_xcorr=mat2cell(GAMMA_xcorr, repmat(gridpoints, [1 count]), n*GAMMA_xcorr_len);
else
    GAMMA_xcorr={};
end
end
```

**v    Netzwerk**

```
%% Main-Datei startet Training eines neuen Netzes, initialisiert Gewichte
% AutorIn: Reinhold Roden reinhold.roden@gmx.net
% basiert auf dem Skript der OXlearn-Toolbox OX_main
%
%
% Update: 16.06.2016

clear
close all
clc
%%
%Art des Signals
signal_mode={'noise' 'speech'};

%mit/ohne Bottleneck
OXnetworkType={'n-layer bottleneck feed-forward' 'n-layer feed-forward'};

%Fensterausschnitt auf Kugel
mode_data_set={'front' 'side'};

% HRTF-Datensatz
data_set='KEMAR_HA';

%Art des Netzwerkausgangs----lernt nur az-azimuth (7 Neuronen),
%nur el-elevation (7 Neuronen), az_el_full-sphärisch (7x7=49 Neuronen)
% az_el_attached-azimuth und elevation angehängt (7+7 Neuronen)
target_mode={'az' 'el' 'az_el_full' 'az_el_attached'};
%%
switch OXnetworkType %Fallunterscheidung Netztyp
    case 'n-layer bottleneck feed-forward'
        net_type='Bottle';
    case 'n-layer feed-forward';
        net_type='noBottle';
end

switch signal_mode %Fallunterscheidung Daten je Art des Signals
    case 'speech'
        filename={'az_el_ILD_ITD_broad_speaker1_filenum100.mat'...
            'az_el_ILD_broad_speaker1_filenum100.mat'...
            'az_el_ITD_broad_speaker1_filenum100.mat'...
            'az_el_mfcc_speaker1_filenum100.mat'...
            'az_el_mag_speaker1_filenum100.mat'...
            'az_el_ILD_narrow_speaker1_filenum100.mat'...
            'az_el_ITD_narrow_speaker1_filenum100.mat'...
            'az_el_ILD_ITD_narrow_speaker1_filenum100.mat'};

    case 'noise'
        filename={'az_el_ILD_ITD_broad_noise.mat'...
            'az_el_ILD_broad_noise.mat'...
            'az_el_ITD_broad_noise.mat'...
            'az_el_mfcc_noise.mat'...
            'az_el_mag_noise.mat'...
            'az_el_ILD_narrow_noise.mat'...
            'az_el_ITD_narrow_noise.mat'...}
```

```
        'az_el_ILD_ITD_narrow_noise.mat'};
end

for aa=1:length(filename) %für alle Datensätze verschiedener Merkmale
    for bb=1:length(target_mode)
        clearvars -except signal_mode mode_data_set data_set target_mode net_type ✓
        OXnetworkType filename aa bb
        disp([net_type ' ' signal_mode ' ' mode_data_set ' ' target_mode{bb} ' ' filename ✓
{aa}])

        switch signal_mode %Eingangsdaten für Netzwerk laden
            case 'speech'
                load(fullfile('home/roderd/TOOLBOX/dnn_input_output', data_set, 'binaural', ✓
mode_data_set, filename{aa}));%, 'OXinput', 'OXtarget', 'OXtestInput', 'OXtestTarget');
            case 'noise'
                load(fullfile('home/roderd/TOOLBOX/dnn_input_output_noise', data_set, ✓
'binaural', mode_data_set, filename{aa}));%, 'OXinput', 'OXtarget', 'OXtestInput', ✓
'OXtestTarget');
        end

        OXsimFilePath=['/home/roderd/TOOLBOX/OXlearn_added/Simulations/TRAINING_' ✓
signal_mode '_az_el/' net_type '/' mode_data_set '/' target_mode{bb} '/'];
        if ~exist(OXsimFilePath, 'dir')
            mkdir(OXsimFilePath) %Speicherort anlegen
        end

        switch target_mode{bb} %Fallunterscheidung Art des Ausgangsvektors
            case 'az_el_full'
                OXtarget=OXtarget_az_el;
                OXtestTarget=OXtestTarget_az_el;
            case 'az_el_attached'
                OXtarget=OXtarget;
                OXtestTarget=OXtestTarget;
            case 'az'
                OXtarget=OXtarget_az;
                OXtestTarget=OXtestTarget_az;
            case 'el'
                OXtarget=OXtarget_el;
                OXtestTarget=OXtestTarget_el;
        end

        if strcmp(net_type, 'Bottle')
            OXtarget2=OXtarget;
            OXtestTarget2=OXtestTarget;
            OXtarget=OXtarget_az_el;
            OXtestTarget=OXtestTarget_az_el;
        end

        clear OXtarget_az_el OXtestTarget_az_el OXtarget_az OXtestTarget_az OXtarget_el ✓
OXtestTarget_el

        n_layer=5; %Ebenenanzahl
        bottle_units=ceil(size(OXinput, 2)/2); %Neuronenanzahl Bottleneck
        bottle_where=4; %Ort des Bottlenecks
        OXlr=0.04; %Lernrate
```

```
OXmomentum=0.5; % 0.5-0.95, Trägheitsterm bzw. Momentum
OXnH=round(1.5*size(OXinput,2)); %Neuronenanzahl versteckter Ebenen
max_epoch=30;%max. Epochenanzahl im Training
min_epoch=10;%min. Epochenanzahl im Training
OXmaxSweeps=size(OXinput, 1)*max_epoch; %max. Iterationenanzahl
OXminSweeps=size(OXinput, 1)*min_epoch; %min. Iterationenanzahl

%weitere Toolboxparameter
OXactFcnH='sigmoid';
OXactFcnO='sigmoid';
OXautoTestFlag=1;
OXautoVerifyFlag=1;
OXdumpEveryNSweeps=0;
OXdumpFlag=0;
OXearlyStoppingFlag=1;
OXfahlmannOffset=0;
OXlearningAlgorithm='backprop';
OXliveUpdateFlag=0;
OXlogTrainPerfFlag=1;
OXlogTrainPerfInterval=1;

OXnI=size(OXinput,2);
OXnO=size(OXtarget,2);
OXbH=1;
OXbO=1;
OXpresentationOrder='random without replacement';
OXseedNr=1;
OXsimFileName=filename{aa};

OXstopCritType='error <=';
OXstopCritValue=1*10^(-8);
OXstopCritWindow=1;
OXtestLogHiddenActFlag=1;
OXtestLogHiddenActFlag=1;
OXwInitType='random seed';
OXwInitRange=0.1;
OXwInitMean=0;

mode='normal';
OXnSweeps=1;
OXtrainError=NaN;
OXtrainCorrect=NaN;
OXtrainOrderLog=NaN;
evalin('base','clear OXverifyOutput OXverifyHidden OXtestOutput OXtestHidden');

%% training
switch evalin('base','OXnetworkType')
case 'n-layer feed-forward'
    %weight initialisation
    tmp= repmat({'OXweightsHwn'}, [1 n_layer]); Hwn=genvarname(tmp(1:n_layer-1)); Hwn=Hwn(2:end);
    tmp= repmat({'OXweightsHwb'}, [1 n_layer]); Hwb=genvarname(tmp(1:n_layer-1)); Hwb=Hwb(2:end);
    initWeights({Hwn{1:end} Hwb{1:end} 'OXweightsHiddenToOutput' 'OXweightsToOutputBias'});
```



```
%run training
OX_trainFFn_wGUI(mode);
case 'n-layer bottleneck feed-forward'
% part I get bottleneck features
%weight initialisation
tmp= repmat({'OXweightsHwn'}, [1 n_layer]); Hwn=genvarname(tmp(1:n_layer-1)); Hwn=Hwn(2:end);
tmp= repmat({'OXweightsHwb'}, [1 n_layer]); Hwb=genvarname(tmp(1:n_layer-1)); Hwb=Hwb(2:end);
initWeights({Hwn{1:end} Hwb{1:end} 'OXweightsBottleToOutput' 'OXweightsHiddenToOutput' 'OXweightsToOutputBias'});

%run training
[OX_bottle_feature_input, OX_bottle_feature_test, OX_bottle_sim_path] = OX_trainFFn_bottle_wGUI(mode);

% part II
OXnetworkType='n-layer feed-forward';
OXinput= OX_bottle_feature_input;
OXnI=size(OXinput,2);
OXnH=round(1.5*size(OXinput,2));
OXtarget=OXtarget2;
OXtestTarget=OXtestTarget2;
OXnO=size(OXtarget,2);
OXtestInput=OX_bottle_feature_test;

%weight initialisation
tmp= repmat({'OXweightsHwn'}, [1 n_layer]); Hwn=genvarname(tmp(1:n_layer-1)); Hwn=Hwn(2:end);
tmp= repmat({'OXweightsHwb'}, [1 n_layer]); Hwb=genvarname(tmp(1:n_layer-1)); Hwb=Hwb(2:end);
initWeights({Hwn{1:end} Hwb{1:end} 'OXweightsHiddenToOutput' 'OXweightsToOutputBias'});

%run training
OX_trainFFn_wGUI(mode);

% part III - reset
OXnetworkType='n-layer bottleneck feed-forward';
otherwise
end
end
end
```

```
function initWeights(varNames)
%create random weight matrices in accordance with the relevant parameters, e.g.;
%the size (OXnI, OXnH, OXnO...)
%the characteristics of the random distribution to draw from (OXwInitRange, ↙
OXwInitMean)
%uses a specific random seed if given, thus enabling to retain the SAME
%random values whenever the same seed is specified
%varNames must be a string or a cell array of strings with the names of the
%weights variables to create

% Update:          Reinhild Roden reINHILD.RODEN@gmx.net 16.06.2016
%                 angepasste Version für MAIN.m

OXnetworkType=getFromBase('OXnetworkType');
if strcmp(OXnetworkType,'n-layer bottleneck feed-forward')
n_layer=getFromBase('n_layer');
bottle_where=getFromBase('bottle_where');
bottle_units=getFromBase('bottle_units');
end

if ischar(varNames)
    varNames={varNames};
end

initvars={'OXwInitType'; 'OXwInitRange'; 'OXwInitMean'; 'OXnI'; 'OXnH'; ↙
'OXnO'; 'OXseedNr'};
successflags=importVarsFromBase(initvars);
if ~all(successflags(1:6))
    initvars(~successflags(1:6))
    error(['failure in weights initialisation because variables were not found']);
end
for i=1:length(varNames)
    if strcmp(varNames{i},'OXweightsInputToHidden')
        x=OXnI;
        y=OXnH;
    elseif strcmp(varNames{i},'OXweightsToHiddenBias')
        x=1;
        y=OXnH;
    elseif strcmp(varNames{i},'OXweightsToHiddenBias_1')
        x=1;
        y=OXnH;
    elseif strcmp(varNames{i},'OXweightsHiddenToHidden_1_2')
        x=OXnH;
        y=OXnH;
    elseif strcmp(varNames{i},'OXweightsToHiddenBias_2')
        x=1;
        y=OXnO;
    elseif strcmp(varNames{i},'OXweightsHiddenToOutput')

        if strcmp(OXnetworkType,'n-layer bottleneck feed-forward') && ↙
n_layer==bottle_where-1
            x=bottle_units;
            y=OXnO;
        else
```

```
x=OXnH;
y=OXnO;
end
elseif strcmp(varNames{i}, 'OXweightsBottleToOutput')

    x=bottle_units;
    y=OXnO;

elseif strcmp(varNames{i}, 'OXweightsToOutputBias')
    x=1;
    y=OXnO;
elseif strcmp(varNames{i}(1:end), 'OXweightsHwn1')

    if strcmp(OXnetworkType, 'n-layer bottleneck feed-forward') && strcmp(
(varNames{i}(10:end), ['Hwn' num2str(bottle_where-1)]))
        x=OXnI;
        y=bottle_units;
    else
        x=OXnI;
        y=OXnH;
    end

elseif strcmp(varNames{i}(1:12), 'OXweightsHwn')

    if strcmp(OXnetworkType, 'n-layer bottleneck feed-forward') && strcmp(
(varNames{i}(10:end), ['Hwn' num2str(bottle_where-1)]))
        x=OXnH;
        y=bottle_units;
    elseif strcmp(OXnetworkType, 'n-layer bottleneck feed-forward') &&
strcmp(varNames{i}(10:end), ['Hwn' num2str(bottle_where)]))
        x=bottle_units;
        y=OXnH;
    else
        x=OXnH;
        y=OXnH;
    end

elseif strcmp(varNames{i}(1:12), 'OXweightsHwb')

    if strcmp(OXnetworkType, 'n-layer bottleneck feed-forward') && strcmp(
(varNames{i}(10:end), ['Hwb' num2str(bottle_where-1)]))
        x=1;
        y=bottle_units;
    else
        x=1;
        y=OXnH;
    end

elseif strcmp(varNames{i}, 'OXweightsContextToHidden')
    x=OXnH;
    y=OXnH;
elseif strcmp(varNames{i}, 'OXweightsInputToOutput')
    x=OXnI;
    y=OXnO;
else
```

```
        warning(['weights variable not recognized; ' varNames{i}])
    end

    if strcmp(OXwInitType,'random seed') || strcmp(OXwInitType,'seed nr:')
        if strcmp(OXwInitType,'seed nr:')
            rand('seed',OXseedNr); %set seed in random number generator
        end

        ws=OXwInitRange*(rand(y,x)-0.5+OXwInitMean);
        assignin('base',varNames{i},ws);

    elseif strcmp(OXwInitType,'use current weights')
        weightsExist=importVarsFromBase(varNames{i});
        if weightsExist
            if all(size(eval(varNames{i}))==[y x])
                %do nothing, i.e. the existing weights are used
            else
                error(['Oxlearn tried to use existing weights but ' varNames{i} ' ✓
has not the required dimensions. Training aborted!']);
            end
        else
            error(['Oxlearn tried to use existing weights but ' varNames{i} ' was ✓
not found. Training aborted!']);
        end
    else
        error(['OXwInitType not recognized, was: ' OXwInitType])
    end
end

if strcmp(OXwInitType,'use current weights')
    assignin('base','OXwInitType','random seed');
    disp('-----')
    disp('existing weights were checked and will be used')
    disp('-----')
else
    disp('-----')
    disp(varNames)
    disp('initialized-----')
end
```

```
function OX_trainFFn_wGUI(mode)
% modiefied version of OX_trainFF3.m for MAIN.m
% by Reinhild Roden reinhild.roden@gmx.net, 12-2013

%import all variables into this functions workspace
successflags=importVarsFromBase('all');
origfilename=OXsimFileName;

OXnSweeps=1;
input=OXinput'; %transposing here makes matrix multiplication in loop easier
target=OXtarget';
epochLength=size(input,2);
epochCount=0;
logCount=ceil(OXnSweeps/OXlogTrainPerfInterval)-1;
skipcount=-1;
stopFlag=0;
newdraw=mod(OXnSweeps,epochLength); %mod result for newdraw, can be ~=1 if training is resumed
updateinterv=min(ceil(OXmaxSweeps/20),1000);
if OXearlyStoppingFlag
    stopcritaccum=ones(1,OXstopCritWindow);
end
OXmomentum_init=OXmomentum;
OXmaxSweeps_init=OXmaxSweeps;
OXstopCritValue_init=OXstopCritValue;

%initialize internal variables used in OX_trainFFn
%n-2 Hiddenlayers -> n=4, 2 hidden layers
OXweightsHwn={};
OXweightsHwb={};
for kk=1:n_layer-2
    deltaOXweightsHwn{kk}=zeros(eval(['size(OXweightsHwn' num2str(kk) ' ')]));
    deltaOXweightsHwb{kk}=zeros(eval(['size(OXweightsHwb' num2str(kk) ' ')]));
    actHidden{kk}= zeros(OXnH,1);
    deltaHidden{kk}=zeros(OXnH,1);
    OXweightsHwn{kk}=eval(['OXweightsHwn' num2str(kk)]);
    OXweightsHwb{kk}=eval(['OXweightsHwb' num2str(kk)]);
end
deltaOXweightsHiddenToOutput=zeros(size(OXweightsHiddenToOutput));
deltaOXweightsToOutputBias=zeros(size(OXweightsToOutputBias));

% min 3 layer (means 1 hidden layer)
count=0;
for ll=[3:n_layer n_layer]
    count=count+1;
    %training network with sigmoid activation function (logistic function)
    OXnSweeps=1;
    stopFlag=0;

    %initialize/expand logging variables
    OXtrainError=[OXtrainError NaN(1,ceil((OXmaxSweeps-OXnSweeps)/OXlogTrainPerfInterval))];
    OXtrainCorrect=[OXtrainCorrect NaN(1,ceil((OXmaxSweeps-OXnSweeps)/OXlogTrainPerfInterval))];
    OXtrainOrderLog=[OXtrainOrderLog NaN(1,ceil((OXmaxSweeps-OXnSweeps)/OXlogTrainPerfInterval))];
```

```

/OXlogTrainPerfInterval));

% reset values
OXmomentum=OXmomentum_init;
OXmaxSweeps=size(OXinput, 1)*max_epoch;
OXstopCritValue=OXstopCritValue_init;
for OXnSweeps=OXnSweeps:OXmaxSweeps;

    % half of momentum after min_epoch
    if OXnSweeps==OXminSweeps
        OX_momentum=OXmomentum/2;
    end

    %-----choose which pattern to present-----
    if strcmp(OXpresentationOrder,'random with replacement')
        pat=ceil(epochLength*rand); %draw pattern
    else
        if strcmp(OXpresentationOrder,'sequential')
            pat=mod(OXnSweeps,epochLength)+1;
        elseif strcmp(OXpresentationOrder,'random without replacement')
            if mod(OXnSweeps,epochLength)==newdraw %if new epoch
                draw=randperm(epochLength); %create new draw (random reordering)
            of patterns in one epoch)
            end
            pat=draw(mod(OXnSweeps,epochLength)+1); %go sequentially through draw
        end
    end

    %-----forward pass-----

    actHidden{1}=1./(1+exp(-(OXweightsHwn{1} * input(:,pat) + OXweightsHwb{1} *
OXbH)));
    for kk=2:ll-2
        actHidden{kk}=1./(1+exp(-(OXweightsHwn{kk} * actHidden{kk-1} + OXweightsHwb
{kk} * OXbH)));
    end
    actOutput = 1./(1+exp(-(OXweightsHiddenToOutput * actHidden{ll-2} +
OXweightsToOutputBias * OXbO)));

    %-----error calculation-----
    deltaOutput = (target(:,pat) - actOutput);

    %-----error backpropagation-----
    deltaHidden{ll-2}=actHidden{ll-2}.* (1 - actHidden{ll-2}) .*
(OXweightsHiddenToOutput * deltaOutput);
    for kk=ll-3:-1:1
        deltaHidden{kk}=actHidden{kk}.* (1 - actHidden{kk}) .* (transpose
(OXweightsHwn{kk+1}) * deltaHidden{kk+1});
    end

    %-----calculate weight adjustments-----
    deltaOXweightsHiddenToOutput = OXlr * (actHidden{ll-2} * deltaOutput)' +
OXmomentum * deltaOXweightsHiddenToOutput;
    deltaOXweightsToOutputBias = OXlr * (OXbO * deltaOutput)' + OXmomentum *

```

```

deltaOXweightsToOutputBias;
    for kk=11-2:-1:2
        deltaOXweightsHwn{kk}=OXlr * (actHidden{kk-1} * transpose(deltaHidden{kk}))' + OXmomentum * deltaOXweightsHwn{kk};
        deltaOXweightsHwb{kk}=OXlr * (OXbH * transpose(deltaHidden{kk}))' + OXmomentum * deltaOXweightsHwb{kk};
    end
    deltaOXweightsHwn{1} = OXlr * (input(:,pat) * deltaHidden{1})' + OXmomentum * deltaOXweightsHwn{1};
    deltaOXweightsHwb{1} = OXlr * (OXbH * deltaHidden{1})' + OXmomentum * deltaOXweightsHwb{1};

    %-----update weights-----
    OXweightsHiddenToOutput = OXweightsHiddenToOutput + deltaOXweightsHiddenToOutput;
    OXweightsToOutputBias = OXweightsToOutputBias + deltaOXweightsToOutputBias;

    for kk=1:11-2
        OXweightsHwn{kk}= OXweightsHwn{kk} + deltaOXweightsHwn{kk};
        OXweightsHwb{kk}= OXweightsHwb{kk} + deltaOXweightsHwb{kk};

        assignin('base', ['OXweightsHwn' num2str(kk)], OXweightsHwn{kk});
        assignin('base', ['OXweightsHwb' num2str(kk)], OXweightsHwb{kk});
    end

    %-----calculate and log performance-----
    if mod(OXnSweeps, OXlogTrainPerfInterval)==0
        logCount=logCount+1;
        OXtrainError(:,logCount)=sum(deltaOutput.^2)/OXnO;
        if ~mod(OXnSweeps,1000)
            disp(['trainCorrect: ' num2str(OXtrainCorrect(:,logCount-1)) ' ']);
            trainError: ' num2str(OXtrainError(:,logCount)) ' ' ' num2str(OXnSweeps/OXmaxSweeps*100) '
            '% done of ' num2str(count) 'th iteration (' num2str(n_layer-1) ' at all)');
        end
        OXtrainCorrect(:,logCount)=max(abs(deltaOutput))<0.1; %TODO: implement
    other criterions
        OXtrainOrderLog(:,logCount)=pat;
    end

    %-----countdown to next pause if skipping-----
    if skipcount>0
        skipcount=skipcount-1;
    end

    %-----criterion for stoping training earlier-----
    if OXstopCritValue>sum(deltaOutput.^2)/OXnO;
        OXstopCritValue=sum(deltaOutput.^2)/OXnO;
    end

    if OXnSweeps==round(0.9*OXmaxSweeps);
        OXstopCritValue=10*min(OXtrainError(1:logCount))
    end

    if OXearlyStoppingFlag==1 && OXnSweeps>=OXminSweeps %check if criterion is met
        if strcmp(OXstopCritType(1:5), 'error')

```



```

stopcritaccum(mod(OXnSweeps,OXstopCritWindow)+1)=sum(deltaOutput.^2) ✓
/OXnO;
    if all(stopcritaccum <= OXstopCritValue)
        stopFlag=1;
        OXmaxSweeps=OXnSweeps;
        OXtrainError=OXtrainError(1:logCount);
        OXtrainCorrect=OXtrainCorrect(1:logCount);
        OXtrainOrderLog=OXtrainOrderLog(1:logCount);
    end
elseif strcmp(OXstopCritType(1:7),'correct')
    stopcritaccum(mod(OXnSweeps,OXstopCritWindow)+1)=max(abs(deltaOutput)) ✓
<0.1;
    if all(stopcritaccum >= OXstopCritValue)
        stopFlag=1;
        OXmaxSweeps=OXnSweeps;
        OXtrainError=OXtrainError(1:logCount);
        OXtrainCorrect=OXtrainCorrect(1:logCount);
        OXtrainOrderLog=OXtrainOrderLog(1:logCount);
    end
end
end

%perform dump, at regular intervals if requested, but always at the
%end of training
if stopFlag || OXnSweeps==OXmaxSweeps || (OXdumpFlag==1 && mod(OXnSweeps, ✓
OXdumpEveryNSweeps)==1)
    %before dumping, perform verify and possibly run a test
    if OXautoVerifyFlag %verify_vec
        [OXverifyOutput, OXverifyHidden]=OX_testFFn_wGUI([],[],'verify',...
        [OXweightsHwn{1:11-2} OXweightsHwb{1:11-2} ...
        {OXweightsHiddenToOutput} {OXweightsToOutputBias}]);
    end
    if OXautoTestFlag
        [OXtestOutput, OXtestHidden]=OX_testFFn_wGUI([],[],'test',...
        [OXweightsHwn{1:11-2} OXweightsHwb{1:11-2} ...
        {OXweightsHiddenToOutput} {OXweightsToOutputBias}]);
    end
    %then save all variables beginning with 'OX' to file (attach
    %number current sweep)
    OXtimeStamp=datestr(clock);
    OX_trained_hidden_layers=11;
    OX_bottle_where=bottle_where;
    OXsimFileName=[origfilename(1:end-4) '_date' datestr(clock, ✓
'yyyymmddTHHMMSS') '.mat'];

    if strcmp(net_type, 'Bottle')
        save([OXsimFilePath OXsimFileName(1:end-4) '_' num2str(count+n_layer-1) ✓
'.mat'],'-regexp', '^OX', '-v7.3');
        disp(['dumping: ' OXsimFileName(1:end-4) '_' num2str(count+n_layer-1) ✓
'.mat'])
    elseif strcmp(net_type, 'noBottle')
        save([OXsimFilePath OXsimFileName(1:end-4) '_' num2str(count) '._ ✓
mat'],'-regexp', '^OX', '-v7.3');
        disp(['dumping: ' OXsimFileName(1:end-4) '_' num2str(count) '.mat'])
    else error(['net_type should be Bottle or noBottle']);
end

```



```
        end
    end

    if mod(OXnSweeps,epochLength)==1      %if new epoch
        epochCount=epochCount+1;
    end

    if stopFlag %stop early
        break;
    end
end %of sweeps loop-----
end %of greedy layerwise loop-----

%export all OXlearn variables back to the basic workspace
OXsimFileName=origfilename;
clear OXtimeStamp;
exportVarList=who('-regexp', '^OX');
for j=1:length(exportVarList)
    assignin('base',exportVarList{j},eval(exportVarList{j}));
end
```

---

```

function [varargout]=OX_trainFFn_bottle_wGUI(mode)
% modiefied version of OX_trainFF3.m for MAIN.m
% by Reinhild Roden reinhild.roden@gmx.net, 12-2013

%import all variables into this functions workspace
successflags=importVarsFromBase('all');
origfilename=OXsimFileName;

input=OXinput'; %transposing here makes matrix multiplication in loop easier
target=OXtarget';
epochLength=size(input,2);
epochCount=0;
logCount=ceil(OXnSweeps/OXlogTrainPerfInterval)-1;
skipcount=-1;
stopFlag=0;
newdraw=mod(OXnSweeps,epochLength); %mod result for newdraw, can be ~=1 if training is resumed
updateinterv=min(ceil(OXmaxSweeps/20),1000);
if OXearlyStoppingFlag
    stopcritaccum=ones(1,OXstopCritWindow);
end
OXmomentum_init=OXmomentum;
OXmaxSweeps_init=OXmaxSweeps;
OXstopCritValue_init=OXstopCritValue;

%initialize internal variables used in OX_trainFFn
%n-2 Hiddenlayers -> n=4, 2 hidden layers

OXweightsHwn={};
OXweightsHwb={};
for kk=1:n_layer-2
    if kk==bottle_where-1;
        nH=bottle_units;
    else
        nH=OXnH;
    end

    deltaOXweightsHwn{kk}=zeros(eval(['size(OXweightsHwn' num2str(kk) ' ')]));
    deltaOXweightsHwb{kk}=zeros(eval(['size(OXweightsHwb' num2str(kk) ' ')]));
    actHidden{kk}= zeros(nH,1);
    deltaHidden{kk}=zeros(nH,1);

    OXweightsHwn{kk}=eval(['OXweightsHwn' num2str(kk)]);
    OXweightsHwb{kk}=eval(['OXweightsHwb' num2str(kk)]);
end
deltaOXweightsHiddenToOutput=zeros(size(OXweightsHiddenToOutput));
deltaOXweightsBottleToOutput=zeros(size(OXweightsBottleToOutput));
deltaOXweightsToOutputBias=zeros(size(OXweightsToOutputBias));

% min 3 layer (means 1 hidden layer)
count=0;
for ll=[3:n_layer n_layer]
    count=count+1;
    %training network with sigmoid activation function (logistic function)
    -----

```

```
OXnSweeps=1;
stopFlag=0;

%initialize/expand logging variables
OXtrainError=[OXtrainError NaN(1,ceil((OXmaxSweeps-OXnSweeps) ✓
/OXlogTrainPerfInterval))];
OXtrainCorrect=[OXtrainCorrect NaN(1,ceil((OXmaxSweeps-OXnSweeps) ✓
/OXlogTrainPerfInterval))];
OXtrainOrderLog=[OXtrainOrderLog NaN(1,ceil((OXmaxSweeps-OXnSweeps) ✓
/OXlogTrainPerfInterval))];

% reset values
OXmomentum=OXmomentum_init;
OXmaxSweeps=size(OXinput, 1)*max_epoch;
OXstopCritValue=OXstopCritValue_init;
for OXnSweeps=OXnSweeps:OXmaxSweeps;
    % half of momentum after min_epoch
    if OXnSweeps==OXminSweeps
        OX_momentum=OXmomentum/2;
    end

    %-----choose which pattern to present-----
    if strcmp(OXpresentationOrder,'random with replacement')
        pat=ceil(epochLength*rand); %draw pattern
    else
        if strcmp(OXpresentationOrder,'sequential')
            pat=mod(OXnSweeps,epochLength)+1;
        elseif strcmp(OXpresentationOrder,'random without replacement')
            if mod(OXnSweeps,epochLength)==newdraw %if new epoch
                draw=randperm(epochLength); %create new draw (random reordering ✓
of patterns in one epoch)
            end
            pat=draw(mod(OXnSweeps,epochLength)+1); %go sequentially through draw
        end
    end

    %-----forward pass-----
    actHidden{1}=1./(1+exp(-(OXweightsHwn{1} * input(:,pat) + OXweightsHwb{1} * ✓
OXbH)));
    for kk=2:ll-2
        actHidden{kk}=1./(1+exp(-(OXweightsHwn{kk} * actHidden{kk-1} + OXweightsHwb{ ✓
{kk} * OXbH)));
    end

    if ll==bottle_where+1
        if OX_bottle_layer_sigmoid==1
            actOutput = 1./(1+exp(-(OXweightsBottleToOutput * actHidden{ll-2} + ✓
OXweightsToOutputBias * OXbO)));
        elseif OX_bottle_layer_sigmoid==0
            actOutput = OXweightsBottleToOutput * actHidden{ll-2} + ✓
OXweightsToOutputBias * OXbO; % without sigmoid fnct. if bottleneck layer
        else
            end
        else
            actOutput = 1./(1+exp(-(OXweightsHiddenToOutput * actHidden{ll-2} + ✓
```

```
OXweightsToOutputBias * OXbO));
end
%-----error calculation-----
deltaOutput = (target(:,pat) - actOutput);

%-----error backpropagation-----
if ll==bottle_where+1
    deltaHidden{ll-2}=actHidden{ll-2}.* (1 - actHidden{ll-2}) .* ✓
(OXweightsBottleToOutput' * deltaOutput);
else
    deltaHidden{ll-2}=actHidden{ll-2}.* (1 - actHidden{ll-2}) .* ✓
(OXweightsHiddenToOutput' * deltaOutput);
end

for kk=ll-3:-1:1
    deltaHidden{kk}=actHidden{kk}.* (1 - actHidden{kk}) .* (transpose ✓
(OXweightsHwn{kk+1}) * deltaHidden{kk+1});
end

%-----calculate weight adjustments-----
if ll==bottle_where+1
    deltaOXweightsBottleToOutput = OXlr * (actHidden{ll-2} * deltaOutput')' + ✓
OXmomentum * deltaOXweightsBottleToOutput;
else
    deltaOXweightsHiddenToOutput = OXlr * (actHidden{ll-2} * deltaOutput')' + ✓
OXmomentum * deltaOXweightsHiddenToOutput;
end
    deltaOXweightsToOutputBias = OXlr * (OXbO * deltaOutput')' + OXmomentum * ✓
deltaOXweightsToOutputBias;

for kk=ll-2:-1:2
    deltaOXweightsHwn{kk}=OXlr * (actHidden{kk-1} * transpose(deltaHidden ✓
{kk}))' + OXmomentum * deltaOXweightsHwn{kk};
    deltaOXweightsHwb{kk}=OXlr * (OXbH * transpose(deltaHidden{kk}))' + ✓
OXmomentum * deltaOXweightsHwb{kk};
end
    deltaOXweightsHwn{1} = OXlr * (input(:,pat) * deltaHidden{1}')' + OXmomentum * ✓
deltaOXweightsHwn{1};
    deltaOXweightsHwb{1} = OXlr * (OXbH * deltaHidden{1}')' + OXmomentum * ✓
deltaOXweightsHwb{1};

%-----update weights-----
if ll==bottle_where+1
    OXweightsBottleToOutput = OXweightsBottleToOutput + ✓
deltaOXweightsBottleToOutput;
else
    OXweightsHiddenToOutput = OXweightsHiddenToOutput + ✓
deltaOXweightsHiddenToOutput;
end
    OXweightsToOutputBias = OXweightsToOutputBias + deltaOXweightsToOutputBias;

for kk=1:ll-2
    OXweightsHwn{kk}= OXweightsHwn{kk} + deltaOXweightsHwn{kk};
    OXweightsHwb{kk}= OXweightsHwb{kk} + deltaOXweightsHwb{kk};
```

```

        assignin('base', ['OXweightsHwn' num2str(kk)], OXweightsHwn{kk});
        assignin('base', ['OXweightsHwb' num2str(kk)], OXweightsHwb{kk});
    end

    %-----calculate and log performance-----
    if mod(OXnSweeps, OXlogTrainPerfInterval)==0
        logCount=logCount+1;
        OXtrainError(:,logCount)=sum(deltaOutput.^2)/OXnO;
        if ~mod(OXnSweeps,1000)
            disp(['trainCorrect: ' num2str(OXtrainCorrect(:,logCount-1)) ' ']);
trainError: ' num2str(OXtrainError(:,logCount)) ' ' ' num2str(OXnSweeps/OXmaxSweeps*100) ✓
'% done of ' num2str(count) 'th iteration (' num2str(n_layer-1) ' at all)');
        end
        OXtrainCorrect(:,logCount)=max(abs(deltaOutput))<0.1; %TODO: implement ✓
    other criterions
        OXtrainOrderLog(:,logCount)=pat;
    end

    %-----countdown to next pause if skipping-----
    if skipcount>0
        skipcount=skipcount-1;
    end

    %-----criterion for stoping training earlier-----
    if OXstopCritValue>sum(deltaOutput.^2)/OXnO;
        OXstopCritValue=sum(deltaOutput.^2)/OXnO
    end

    if OXnSweeps==round(0.9*OXmaxSweeps);
        OXstopCritValue=10*min(OXtrainError(1:logCount))
    end

    if OXearlyStoppingFlag==1 && OXnSweeps>=OXminSweeps %check if criterion is met
        if strcmp(OXstopCritType(1:5), 'error')
            stopcritaccum(mod(OXnSweeps,OXstopCritWindow)+1)=sum(deltaOutput.^2) ✓
/OXnO;
            if all(stopcritaccum <= OXstopCritValue)
                stopFlag=1;
                OXmaxSweeps=OXnSweeps;
                OXtrainError=OXtrainError(1:logCount);
                OXtrainCorrect=OXtrainCorrect(1:logCount);
                OXtrainOrderLog=OXtrainOrderLog(1:logCount);
            end
        elseif strcmp(OXstopCritType(1:7), 'correct')
            stopcritaccum(mod(OXnSweeps,OXstopCritWindow)+1)=max(abs(deltaOutput)) ✓
<0.1;
            if all(stopcritaccum >= OXstopCritValue)
                stopFlag=1;
                OXmaxSweeps=OXnSweeps;
                OXtrainError=OXtrainError(1:logCount);
                OXtrainCorrect=OXtrainCorrect(1:logCount);
                OXtrainOrderLog=OXtrainOrderLog(1:logCount);
            end
        end
    end
end
end

```

```

        %perform dump, at regular intervals if requested, but always at the
        %end of training
        if stopFlag || OXnSweeps==OXmaxSweeps || (OXdumpFlag==1 && mod(OXnSweeps,
OXdumpEveryNSweeps)==1)
            %before dumping, perform verify and possibly run a test
            if OXautoVerifyFlag %verify_vec
                if ll==bottle_where+1
                    [OXverifyOutput, OXverifyHidden]=OX_testFFn_bottle_wGUI([],
[], 'verify', ...
                    [OXweightsHwn{1:ll-2} OXweightsHwb{1:ll-2} ...
                    {OXweightsBottleToOutput} {OXweightsToOutputBias}]);
                else
                    [OXverifyOutput, OXverifyHidden]=OX_testFFn_bottle_wGUI([],
[], 'verify', ...
                    [OXweightsHwn{1:ll-2} OXweightsHwb{1:ll-2} ...
                    {OXweightsHiddenToOutput} {OXweightsToOutputBias}]);
                end
            end
            if OXautoTestFlag
                if ll==bottle_where+1
                    [OXtestOutput, OXtestHidden]=OX_testFFn_bottle_wGUI([],
[], 'test', ...
                    [OXweightsHwn{1:ll-2} OXweightsHwb{1:ll-2} ...
                    {OXweightsBottleToOutput} {OXweightsToOutputBias}]);
                else
                    [OXtestOutput, OXtestHidden]=OX_testFFn_bottle_wGUI([],
[], 'test', ...
                    [OXweightsHwn{1:ll-2} OXweightsHwb{1:ll-2} ...
                    {OXweightsHiddenToOutput} {OXweightsToOutputBias}]);
                end
            end
            %then save all variables beginning with 'OX' to file (attach
            %number current sweep)
            OXtimeStamp=datestr(clock);
            OX_trained_hidden_layers=ll-2;
            OX_bottle_where=bottle_where;
            OXsimFileName=[origfilename(1:end-4) '_date' datestr(clock,
'yyyymmddTHHMMSS') '.mat'];
            save([OXsimFilePath OXsimFileName(1:end-4) '_' num2str(count) '.mat'], '-
regexp', '^OX', '-v7.3');
            disp(['dumping: ' OXsimFileName(1:end-4) '_' num2str(count) '.mat'])
        end

        if mod(OXnSweeps,epochLength)==1 %if new epoch
            epochCount=epochCount+1;
        end

        if stopFlag %stop early
            break;
        end
    end %of sweeps loop-----
end %of greedy layerwise loop-----

%% give bottleneck features for input vector

```

```
if ll==bottle_where+1

    [OXtestOutput_input, OXtestHidden_input]=OX_testFFn_bottle_wGUI([],[],'verify',...
        [OXweightsHwn{1:ll-2} OXweightsHwb{1:ll-2} ...
        {OXweightsBottleToOutput} {OXweightsToOutputBias}]);

    [OXtestOutput_test, OXtestHidden_test]=OX_testFFn_bottle_wGUI([],[],'test',...
        [OXweightsHwn{1:ll-2} OXweightsHwb{1:ll-2} ...
        {OXweightsBottleToOutput} {OXweightsToOutputBias}]);

else

    [OXtestOutput_input, OXtestHidden_input]=OX_testFFn_bottle_wGUI([],[],'verify',...
        [OXweightsHwn{1:ll-2} OXweightsHwb{1:ll-2} ...
        {OXweightsHiddenToOutput} {OXweightsToOutputBias}]);

    [OXtestOutput_test, OXtestHidden_test]=OX_testFFn_bottle_wGUI([],[],'test',...
        [OXweightsHwn{1:ll-2} OXweightsHwb{1:ll-2} ...
        {OXweightsBottleToOutput} {OXweightsToOutputBias}]);

end
varargout{1}=OXtestHidden_input{bottle_where-1};
varargout{2}=OXtestHidden_test{bottle_where-1};
varargout{3}=[OXsimFilePath OXsimFileName(1:end-4) '_' num2str(count) '.mat'];

%export all OXlearn variables back to the basic workspace
OXsimFileName=origfilename;
OXstopCritValue=OXstopCritValue_init;
clear OXtimeStamp OXstopCritValue_init

exportVarList=who('-regexp', '^OX');
for j=1:length(exportVarList)
    assignin('base',exportVarList{j},eval(exportVarList{j}));
end
end
```

```
function varargout=OX_testFFn_wGui(a,b,type,weights)
% modiefied version of OX_testFF3.m for MAIN.m
% by Reinhild Roden reinhild.rodén@gmx.net, 12-2013

%list all the variables that we need
varsList={'OXnI';
          'OXnH';
          'OXnO';
          'OXbH';
          'OXbO';
          'OXweightsHiddenToOutput';
          'OXweightsToOutputBias';
          'OXinput';
          'OXtarget';
          'OXtestInput';
          'OXtestTarget'};

%import the variables on the list into this functions workspace
successflags=importVarsFromBase(varsList);

if ~all(successflags)
    if successflags(end-1)==0 && strcmp(type,'verify')
        %don't need testInput if only verifying
    else
        disp(char(varsList{~successflags}))
        error(['failure to test network because these variables were not found']);
    end
end

%if weights are send as argument (e.g. from training), use these instead of the ones in
the base work space
if nargin==4 && ~isempty(weights) %&& length(weights)==4
    ind=(length(weights)-2)/2;
    OXweightsHwn={weights{1:ind}};
    OXweightsHwb={weights{ind+1:2*ind}};
    OXweightsHiddenToOutput={weights{end-1}};
    OXweightsToOutputBias={weights{end}};
end

if strcmp(type,'verify')
    OXtestInput=OXinput'; %transposing here makes matrix multiplication in loop easier
elseif strcmp(type,'test')
    OXtestInput=OXtestInput';
else
    error('test type not recognized');
end

maxSweeps=size(OXtestInput,2);
epochCount=0;
OXbH=ones(1,maxSweeps)*OXbH;
OXbO=ones(1,maxSweeps)*OXbO;

OXtestHidden={};

%testing network with sigmoid activation function
```



```
%-----forward pass-----
curr_num_hidden_layer=size(OXweightsHwn,2);
OXtestHidden{1}=1./(1+exp(-(OXweightsHwn{1} * OXtestInput + OXweightsHwb{1} * OXbH)));
for kk=2:curr_num_hidden_layer
    OXtestHidden{kk}=1./(1+exp(-(OXweightsHwn{kk} * OXtestHidden{kk-1} + OXweightsHwb{kk} * OXbH)));
end

if isempty(kk)
    kk=1;
end
OXtestOutput = 1./(1+exp(-(OXweightsHiddenToOutput{1} * OXtestHidden{kk} + OXweightsToOutputBias{1} * OXbO)));

%export Output and Hidden, if requested
if nargin==2
    %varargout{1}=OXtestOutput';
    varargout=[OXtestOutput' {cellfun(@transpose,OXtestHidden,'UniformOutput',false)}];
elseif strcmp(type,'verify')
    assignin('base','OXverifyOutput',OXtestOutput');
    assignin('base','OXverifyHidden',cellfun(@transpose,OXtestHidden,'UniformOutput',false));
elseif strcmp(type,'test')
    assignin('base','OXtestOutput',OXtestOutput');
    assignin('base','OXtestHidden',cellfun(@transpose,OXtestHidden,'UniformOutput',false));
end
```

## vi Evaluation

```
% script evaluates neural network by testing it.
%
% Reinhild Roden, reinhild.roden@gmx.net, 02/2015
clc
clear
close all

data_set='KEMAR_HA';
signal_mode={'noise' 'speech'};
mode_data_set={'front' 'side'};
target_mode={'az' 'el' 'az_el_full' 'az_el_attached'};
net_type={'Bottle' 'noBottle'};
n_layer=5;
num_mean=10;

OXinput=[];
OXtarget=[];
OXinput=[];

% loop signal mode - noise or speech
for aa=1:length(signal_mode)

    % loop mode - front or side
    for bb=1:length(mode_data_set)

        % loop target mode - az el full attached
        for cc=1:length(target_mode)

            % loop network type - bottleneck or no bottleneck
            for dd=1:length(net_type)

                % find simulation files
                sim_path=fullfile('OXlearn_added','Simulations', ['TRAINING_' signal_mode{aa}
{aa} '_az_el'],...
                    net_type{dd}, mode_data_set{bb} , target_mode{cc});
                files_sim = dir(sim_path);
                filename = {files_sim.name};
                filename(1:2) = [];

                % path for evaluation results
                save_path=fullfile(['eval_test_' signal_mode{aa} '_az_el'], ...
                    net_type{dd}, mode_data_set{bb}, target_mode{cc});
                if ~exist(save_path, 'dir')
                    switch net_type{dd}
                        case 'Bottle'
                            mkdir([save_path '/sig0/png'])
                            mkdir([save_path '/sig0/fig'])
                            mkdir([save_path '/sig0/mat'])

                            mkdir([save_path '/sig1/png'])
                            mkdir([save_path '/sig1/fig'])
                            mkdir([save_path '/sig1/mat'])
                        case 'noBottle'
                            mkdir([save_path '/png'])
                            mkdir([save_path '/fig'])
```

```

        mkdir([save_path '/mat'])
    end
end

% loop files - all simulated conditions, all trained networks
for ee=1:length(filename)
    disp([filename{ee} ' ' mode_data_set{bb}])

    % load test-set
    if ~isempty(strfind(filename{ee}, 'az_el_ILD_broad'))
        test_file=['az_el_ILD_broad_' signal_mode{aa}];

    elseif ~isempty(strfind(filename{ee}, 'az_el_ITD_broad'))
        test_file=['az_el_ITD_broad_' signal_mode{aa}];

    elseif ~isempty(strfind(filename{ee}, 'az_el_ILD_ITD_broad'))
        test_file=['az_el_ILD_ITD_broad_' signal_mode{aa}];

    elseif ~isempty(strfind(filename{ee}, 'az_el_ILD_narrow'))
        test_file=['az_el_ILD_narrow_' signal_mode{aa}];

    elseif ~isempty(strfind(filename{ee}, 'az_el_ITD_narrow'))
        test_file=['az_el_ITD_narrow_' signal_mode{aa}];

    elseif ~isempty(strfind(filename{ee}, 'az_el_ILD_ITD_narrow'))
        test_file=['az_el_ILD_ITD_narrow_' signal_mode{aa}];

    elseif ~isempty(strfind(filename{ee}, 'az_el_mag'))
        test_file=['az_el_mag_' signal_mode{aa}];

    elseif ~isempty(strfind(filename{ee}, 'az_el_mfcc'))
        test_file=['az_el_mfcc_' signal_mode{aa}];
    else
        error('error')
    end
    disp(test_file)

    load(fullfile(['dnn_input_output_' signal_mode{aa} '_test'], data_set,
'binaural',...
        mode_data_set{bb},test_file),'OXtestInput',...
        'OXtestTarget',
'OXtestTarget_az','OXtestTarget_el','OXtestTarget_az_el');

    OXtestTarget_az_el_attached=OXtestTarget;
    OXtestTarget_az_el_full=OXtestTarget_az_el;

    clear OXtestTarget OXtestTarget_az_el

    switch net_type{dd}
    case 'noBottle'
        % load weights and parameters of the neural network to be tested
        load([sim_path '/' filename{ee}], 'OXnI','OXnH','OXnO','OXbH',...
        'OXbO','OXweightsHwn','OXweightsHwb','OXweightsHiddenToOutput',...
        'OXweightsToOutputBias',

```

```

'OX_bottle_where','OX_trained_hidden_layers',...
    'OX_num_hidden_neurons_double_input','OXtimeStamp');

% take specific targets
OXtestTarget=evalin('base',['OXtestTarget_' target_mode(cc)]);

[OXtestOutput, OXtestHidden]=OX_testFFn_wGUI([],[],'test',...
[OXweightsHwn{1:OX_trained_hidden_layers} OXweightsHwb{1:
OX_trained_hidden_layers} ...
    {OXweightsHiddenToOutput} {OXweightsToOutputBias}}');
case 'Bottle'
    load([sim_path '/' filename{ee}], 'OX_bottle_sim_path');
    if ~exist('OX_bottle_sim_path', 'var')
        load([sim_path '/' filename{ee}],
'OXnI','OXnH','OXnO','OXbH',...
'OXbO','OXweightsHwn','OXweightsHwb','OXweightsBottleToOutput','OXweightsHiddenToOutput
',...
    'OXweightsToOutputBias',
'OX_bottle_where','OX_trained_hidden_layers', 'OX_bottle_layer_sigmoid');
    bottle_where=OX_bottle_where;
    % take specific targets
    OXtestTarget=evalin('base','OXtestTarget_az_el_full');

    if OX_trained_hidden_layers==(OX_bottle_where-1)
        [OXtestOutput, OXtestHidden]=OX_testFFn_bottle_wGUI([],
[],'test',...
[OXweightsHwn{1:OX_trained_hidden_layers} OXweightsHwb{1:
OX_trained_hidden_layers} ...
    {OXweightsBottleToOutput} {OXweightsToOutputBias}}');
    else
        [OXtestOutput, OXtestHidden]=OX_testFFn_bottle_wGUI([],
[],'test',...
[OXweightsHwn{1:OX_trained_hidden_layers} OXweightsHwb{1:
OX_trained_hidden_layers} ...
    {OXweightsHiddenToOutput} {OXweightsToOutputBias}}');
    end
elseif exist('OX_bottle_sim_path', 'var')
    %load last bottle-neck trained network
    load(OX_bottle_sim_path, 'OXnI','OXnH','OXnO','OXbH',...
'OXbO','OXweightsHwn','OXweightsHwb','OXweightsBottleToOutput','OXweightsHiddenToOutput
',...
    'OXweightsToOutputBias',
'OX_bottle_where','OX_trained_hidden_layers', 'OX_bottle_layer_sigmoid');
    bottle_where=OX_bottle_where;

% take specific targets
OXtestTarget=evalin('base','OXtestTarget_az_el_full');

if OX_trained_hidden_layers==(OX_bottle_where-1)
    [OXtestOutput, OXtestHidden]=OX_testFFn_bottle_wGUI([],
[],'test',...

```

```

                                [OXweightsHwn{1:OX_trained_hidden_layers} OXweightsHwb{1:✓
OX_trained_hidden_layers} ...
                                {OXweightsBottleToOutput} {OXweightsToOutputBias}}');

                                else
                                [OXtestOutput, OXtestHidden]=OX_testFFn_bottle_wGUI([], ✓
[], 'test', ...
                                [OXweightsHwn{1:OX_trained_hidden_layers} OXweightsHwb{1:✓
OX_trained_hidden_layers} ...
                                {OXweightsHiddenToOutput} {OXweightsToOutputBias}}');
                                end

                                OXtestInput=OXtestHidden{bottle_where-1};
                                OXtestTarget=evalin('base', ['OXtestTarget_' target_mode{cc}]);

                                load([sim_path '/' filename{ee}], ✓
'OXnI', 'OXnH', 'OXnO', 'OXbH', ...
✓
'OXbO', 'OXweightsHwn', 'OXweightsHwb', 'OXweightsBottleToOutput', 'OXweightsHiddenToOutput' ✓
', ...
                                'OXweightsToOutputBias', ✓
'OX_bottle_where', 'OX_trained_hidden_layers', 'OX_bottle_layer_sigmoid');
                                bottle_where=OX_bottle_where;

                                [OXtestOutput, OXtestHidden]=OX_testFFn_wGUI([], [], 'test', ...
                                [OXweightsHwn{1:bottle_where-1} OXweightsHwb{1:bottle_where-1} ✓
...
                                {OXweightsHiddenToOutput} {OXweightsToOutputBias}}');
                                else
                                end
                                end

                                %% testing for used target vector

                                % each frame per frame
                                [ mat_perframe ] = eval_per_frame( OXtestTarget, OXtestOutput );

                                % mean 10 frames
                                [mat_mean, mean_val] = eval_mean_frame( OXtestTarget, OXtestOutput, ✓
num_mean );

                                myFigure(8,5)
                                colormap(gray);colormap(flipud(colormap));

                                h=imagesc(mat_mean); colorbar
                                axis off
                                caxis([0 100])

                                switch net_type{dd}
                                case 'Bottle'
                                if OX_bottle_layer_sigmoid==1
                                saveas(h, fullfile(save_path, 'sig1/png', [filename{ee}(1:end-4) ✓
'.png']), 'png')
                                saveas(h, fullfile(save_path, 'sig1/fig', [filename{ee}(1:end-4) ✓
'.fig']), 'fig')

```

```

elseif OX_bottle_layer_sigmoid==0
    saveas(h, fullfile(save_path,'sig0/png', [filename{ee}(1:end-4)
'.png'])), 'png')
    saveas(h, fullfile(save_path,'sig0/fig', [filename{ee}(1:end-4)
'.fig'])), 'fig')
else
    error('error');
end
case 'noBottle'
    saveas(h, fullfile(save_path,'/png', [filename{ee}(1:end-4)
png'])), 'png')
    saveas(h, fullfile(save_path,'/fig', [filename{ee}(1:end-4)
fig'])), 'fig')
end

% mean
mat_mean_all=[];
for nn=1:size(OXtestTarget,2)
    ind=find(OXtestTarget(:,nn)==1);
    mat_mean_all(nn,:)=mean(OXtestOutput(ind,:),1);
    ind_len(nn)=length(ind);
end

%
switch net_type{dd}
case 'Bottle'
    if OX_bottle_layer_sigmoid==1
        save(fullfile(save_path,'sig1/mat', [filename{
{ee}}],'mat_perframe', 'mat_mean', 'mat_mean_all', 'mean_val','-regex', '^OX', '-v7.
3'));
    elseif OX_bottle_layer_sigmoid==0
        save(fullfile(save_path,'sig0/mat', [filename{
{ee}}],'mat_perframe', 'mat_mean', 'mat_mean_all', 'mean_val','-regex', '^OX', '-v7.
3'));
    else
        error('error');
    end
case 'noBottle'
    save(fullfile(save_path,'/mat', [filename{ee}]),'mat_perframe',
'mat_mean', 'mat_mean_all', 'mean_val','-regex', '^OX', '-v7.3');

end
clear 'OX_bottle_sim_path'
close all

end % for length(filename)
end % for length(net_type)
end % for length(target_mode)
end %for length(mode_data_set)
end % for length(signal_mode)

%% comparable results...
% have to transform az-, el-, atatched-targets into full-targets
eval_mode={'az_el_independent' 'az_el_full' 'az_el_attached'}; %
target_mode={'az' 'az_el_full' 'az_el_attached'}; %'az'

```

```
%% loop signal mode - noise or speech
for aa=1:length(signal_mode)

    % loop mode - front or side
    for bb=1:length(mode_data_set)

        % loop target mode - az el full attached
        for cc=1:length(target_mode)

            % loop network type - bottleneck or no bottleneck
            for dd=1:length(net_type)
                switch net_type{dd}
                    case 'Bottle'
                        sig_type={'sig0', 'sig1'};
                    case 'noBottle'
                        sig_type={''};
                end

                for ff=1:length(sig_type)
                    % find evaluation files as already evaluated for each dimension
                    switch net_type{dd}
                        case 'Bottle'
                            eval_path=fullfile(['eval_test_' signal_mode{aa} '_az_el'], ...
                                net_type{dd}, mode_data_set{bb}, target_mode{cc}, sig_type{ff},
                                'mat');
                        case 'noBottle'
                            eval_path=fullfile(['eval_test_' signal_mode{aa} '_az_el'], ...
                                net_type{dd}, mode_data_set{bb}, target_mode{cc}, 'mat');
                    end

                    files_eval      = dir(eval_path);
                    filename        = {files_eval.name};
                    filename(1:2)   = [];

                    if strcmp(target_mode{cc}, 'az')

                        switch net_type{dd}
                            case 'noBottle'
                                el_path=fullfile(['eval_test_' signal_mode{aa} '_az_el'], ...
                                    net_type{dd}, mode_data_set{bb}, 'el', 'mat');
                            case 'Bottle'
                                el_path=fullfile(['eval_test_' signal_mode{aa} '_az_el'], ...
                                    net_type{dd}, mode_data_set{bb}, 'el', sig_type{ff}, 'mat');
                        end

                        files_el      = dir(el_path);
                        filename_el    = {files_el.name};
                        filename_el(1:2) = [];
```



```

end

% path for evaluation results
switch net_type{dd}
case 'noBottle'
    save_path=fullfile(['eval_test_COMP_' signal_mode{aa} '_az_el'], ...
        net_type{dd}, mode_data_set{bb}, target_mode{cc}, sig_type{ff});
case 'Bottle'
    save_path=fullfile(['eval_test_COMP_' signal_mode{aa} '_az_el'], ...
        net_type{dd}, mode_data_set{bb}, target_mode{cc});
end

if ~exist(save_path, 'dir')
    mkdir([save_path '/png'])
    mkdir([save_path '/fig'])
    mkdir([save_path '/mat'])
end

% loop files - all simulated conditions, all trained networks
for ee=1:length(filename)
    ind_start_num=strfind(filename{ee}, '_'); ind_start_num=ind_start_num(
(end)+1;

    ind_end_num=strfind(filename{ee}, '.'); ind_end_num=ind_end_num(end)-1;
    if str2double(filename{ee}(ind_start_num:ind_end_num))>=n_layer ||
strcmp(net_type{dd}, 'noBottle')
        disp(fullfile(eval_path, [filename{ee}]));
        switch target_mode{cc}
        case 'az' %parallel case
            load(fullfile(eval_path, [filename{
ee}], 'OXtestTarget', 'OXtestOutput', 'OXtestTarget_az_el_full',
'OX_bottle_layer_sigmoid'));
            OXtestTarget_az=OXtestTarget;
            OXtestOutput_az=OXtestOutput;
            OXtestTarget_az_el_full_az=OXtestTarget_az_el_full;

            if strcmp(net_type{dd}, 'Bottle')
                OX_bottle_layer_sigmoid_az=OX_bottle_layer_sigmoid;
            end
            % find el-equivalent for 'az'

            ind=strfind(filename{ee}, 'date');
            until_val=length(filename_el);

            kk=0;
            while kk<until_val
                kk=kk+1;
                if strcmp(filename{ee}(1:ind), filename{kk}(1:ind)) &&
strcmp(filename{kk}(ind_start_num:ind_end_num), filename{ee}(ind_start_num:ind_end_num))

                    if strcmp(net_type{dd}, 'Bottle')
                        load([el_path '/' filename_el{kk}],
'OX_bottle_layer_sigmoid');

                        OX_bottle_layer_sigmoid_el=OX_bottle_layer_sigmoid;
                    elseif strcmp(net_type{dd}, 'noBottle')
                        OX_bottle_layer_sigmoid_az=999;
                        OX_bottle_layer_sigmoid_el=999;
                    end
                end
            end
        end
    end
end

```

```
else
    error('error')
end

if OX_bottle_layer_sigmoid_az==OX_bottle_layer_sigmoid_el
    disp([el_path '/' filename_el{kk}])

    load([el_path '/' filename_el{
{kk}}, 'OXtestTarget', 'OXtestOutput', 'OXtestTarget_az_el_full', '
'OX_bottle_layer_sigmoid']);

    OXtestTarget_el=OXtestTarget;
    OXtestOutput_el=OXtestOutput;
    OXtestTarget_az_el_full_el=OXtestTarget_az_el_full;

    %for validation purposes
    if ~(mean(mean(OXtestTarget_az_el_full_el-
OXtestTarget_az_el_full_az))==0)
        error('not same target vectors')
    end

    kk=until_val;
end
end
end

% merge test Output to one matrix
[~,ind_az]=max(OXtestOutput_az,[],2);
[~,ind_el]=max(OXtestOutput_el,[],2);

ind_full=(ind_az-1).*size(OXtestOutput_az,2)+ind_el;

% matlab numbering e.g. A = 1 4 7 10 13
%                        2 5 8 11 14
%                        3 6 9 12 15
% A matrix 3x5 (MxN)
% entry A(3,2) same entry as A(6)
% entry A(m,n) same entry as A( M*N - M*(N-n) - (M-m) )
%                        3*5    3*(5-2) - (3-3)
N= repmat(size(OXtestTarget_az_el_full,2), [size
(OXtestTarget_az_el_full,1) 1]);
M= repmat(size(OXtestTarget_az_el_full,1), [size
(OXtestTarget_az_el_full,1) 1]);
n=ind_full;
m=(1:size(OXtestTarget_az_el_full,1))';
entry_num=M.*N - M.*(N-n) - (M-m);

OXtestOutput_az_el_full=zeros(size(OXtestTarget_az_el_full));
OXtestOutput_az_el_full(entry_num)=1;

OXtestOutput=OXtestOutput_az_el_full;
OXtestTarget=OXtestTarget_az_el_full;
```

```
case 'az_el_full' % spherical
    load(fullfile(eval_path, [filename
{ee}]), 'OXtestTarget', 'OXtestOutput');
    OXtestTarget_az_el_full=OXtestTarget;

    % no merging necessary but treat matrix like in the other cases
    [~,ind_full]=max(OXtestOutput,[],2);

    % matlab numbering e.g. A = 1 4 7 10 13
    %                               2 5 8 11 14
    %                               3 6 9 12 15
    % A matrix 3x5 (MxN)
    % entry A(3,2) same entry as A(6)
    % entry A(m,n) same entry as A( M*N - M*(N-n) - (M-m) )
    %                               3*5   3*(5-2) - (3-3)
    N= repmat(size(OXtestTarget_az_el_full,2), [size
(OXtestTarget_az_el_full,1) 1]);
    M= repmat(size(OXtestTarget_az_el_full,1), [size
(OXtestTarget_az_el_full,1) 1]);
    n=ind_full;
    m=(1:size(OXtestTarget_az_el_full,1))';
    entry_num=M.*N - M.*(N-n) - (M-m);

    OXtestOutput_az_el_full=zeros(size(OXtestTarget_az_el_full));
    OXtestOutput_az_el_full(entry_num)=1;

    OXtestOutput=OXtestOutput_az_el_full;
    OXtestTarget=OXtestTarget_az_el_full;

case 'az_el_attached' %attached
    load(fullfile(eval_path, [filename
{ee}]), 'OXtestTarget', 'OXtestOutput', 'OXtestTarget_az_el_full');
    OXtestTarget_az=OXtestTarget(:,1:size(OXtestTarget,2)/2);
    OXtestOutput_az=OXtestOutput(:,1:size(OXtestTarget,2)/2);
    OXtestTarget_el=OXtestTarget(:,size(OXtestTarget,2)/2+1:end);
    OXtestOutput_el=OXtestOutput(:,size(OXtestTarget,2)/2+1:end);

    % merge test Output to one matrix
    [~,ind_az]=max(OXtestOutput_az,[],2);
    [~,ind_el]=max(OXtestOutput_el,[],2);

    ind_full=(ind_az-1).*size(OXtestOutput_az,2)+ind_el;

    % matlab numbering e.g. A = 1 4 7 10 13
    %                               2 5 8 11 14
    %                               3 6 9 12 15
    % A matrix 3x5 (MxN)
    % entry A(3,2) same entry as A(6)
    % entry A(m,n) same entry as A( M*N - M*(N-n) - (M-m) )
    %                               3*5   3*(5-2) - (3-3)
```

```

        N= repmat(size(OXtestTarget_az_el_full,2), [size
(OXtestTarget_az_el_full,1) 1]);
        M= repmat(size(OXtestTarget_az_el_full,1), [size
(OXtestTarget_az_el_full,1) 1]);
        n=ind_full;
        m=(1:size(OXtestTarget_az_el_full,1))';
        entry_num=M.*N - M.*(N-n) - (M-m);

        OXtestOutput_az_el_full=zeros(size(OXtestTarget_az_el_full));
        OXtestOutput_az_el_full(entry_num)=1;

        OXtestOutput=OXtestOutput_az_el_full;
        OXtestTarget=OXtestTarget_az_el_full;

end

%% testing and saving

% each frame per frame
[ mat_perframe ] = eval_per_frame( OXtestTarget, OXtestOutput );

% mean 10 frames
[mat_mean, mean_val] = eval_mean_frame( OXtestTarget, OXtestOutput,
num_mean );

myFigure(8,5)
colormap(gray);colormap(flipud(colormap));

h=imagesc(mat_mean); colorbar
axis off
caxis([0 100])
saveas(h, fullfile(save_path, '/png', [filename{ee}(1:end-4) '.
png'])), 'png')

saveas(h, fullfile(save_path, '/fig', [filename{ee}(1:end-4) '.
fig'])), 'fig')

% mean
mat_mean_all=[];
for nn=1:size(OXtestTarget,2)
    ind=find(OXtestTarget(:,nn)==1);
    mat_mean_all(nn,:)=mean(OXtestOutput(ind,:),1);
    ind_len(nn)=length(ind);
end
%
save(fullfile(save_path, '/mat', [filename{ee}]),'mat_perframe',

```

```
'mat_mean', 'mat_mean_all', 'mean_val', '-regexp', '^OX', '-v7.3');  
    clear 'OX_bottle_sim_path'  
    close all  
  
        end  
    end  
end  
end  
end  
end  
end  
end
```

```
function [ mat_mean, mean_val ] = eval_mean_frame(OXtestTarget, OXtestOutput, num_mean)

% function calculates the mean of e.g. 10 frames (num_mean=10)
% input:
% OXtestTarget - target of test data
% OXtestOutput - output of testing
% num_mean - number of vectors summed up
% output:
% mat_mean - confusion matrix
% mean_val - hit rate
%
%initialise
mat_mean=zeros(size(OXtestTarget,2));
ind_len=zeros(1,size(OXtestTarget,2));
mean_val=0;
for nn=1:size(OXtestTarget,2)
    %find direction index coded in a target vector per each single column
    ind=find(OXtestTarget(:,nn)==1);
    OXtestOutput_mean=[];
    count=0;
    % mean of vectors
    for jj=1:num_mean:length(ind)-num_mean
        count=count+1;
        OXtestOutput_mean(count,:)= mean(OXtestOutput(ind(jj:jj+num_mean-1),:),1);
    end

    % calc mean of last one
    num_last=mod(length(ind),num_mean);
    OXtestOutput_mean(count+1,:)= mean(OXtestOutput(ind(num_last+1:end),:),1);

    % calc confusion matrix
    [OutMax, ind_max]=max(OXtestOutput_mean,[],2);
    for kk=1:length(ind_max)
        mat_mean(nn, ind_max(kk))=mat_mean(nn, ind_max(kk))+1;
    end
    ind_len(nn)=count+1;
    mat_mean(nn,:)=mat_mean(nn,:);
    mat_mean(nn,:)=mat_mean(nn,:)./ind_len(nn).*100;
    mean_val=mean_val+mat_mean(nn,nn);
end

% mean hit rate
mean_val=mean_val/nn;
end
```

## Anhang D

# Datenträger

### **i example\_input\_data**

**a noise – front/side**

**b speech – front/side**

### **ii HRTF**

**a HRTF\_short**

**b HRTF\_original\_Thiemann**

### **iii logatom\_used\_data**

### **iv noises**

### **v sourcecode**

Enthält Unterordner und selbst programmierte mat-Files.

### **a OXlearn\_added**

Die Toolbox, wie sie zum Training und Testen für diese Arbeit verwendet wurde.

### **b OXlearn\_added\_files**

Enthält nur zusätzlich programmierte mat-Files zur besseren Übersicht.

### **c OXlearn1.1\_original**

Original-Toolbox.

### **d tools**

Enthält weitere Skripte, die verwendet wurden, zum Teil selbst programmiert.

- vi Arbeit im PDF-Format (hyperref)
- vii Arbeit im PDF-Format (Leseformat)